Premios para los lectores • Estrategia • C y Ensamblador • Aventura • DIV 2 • Arcade

DIV manía

www.prensatecnica.com

Año 2 • Número 7

995 ptas.

PORTUGAL 990 ESC (CONT) 5,98 €

En el CD-Rom
Poser
Bryce 2
Fénix

ESTE MES nuevas SECCIONES

• DIV interno
¿Qué es Fénix?

• Poser
Animando figuras para tus juegos

• Open GL
Maneja todo tipo de gráficos

• Aventuras
Perfecciona los diálogos de tus juegos

• Webs DIV
Divworld, más revistas electrónicas sobre DIV

• DIV 2
Casos prácticos para DIV 2

• RPG
El Rol desde varios puntos de vista

• Share Música
Formatos musicales para DIV 2

Prens@ Técnic

8 413042 087533 00007

La esencia de los videojuegos

CREA LOS JUEGOS MÁS CAÑEROS

Consultoría • Mantenimiento • Distribución • Formación • Programación a medida • Certificación • Material didáctico • Cursos para empresas

Universe Linux™

La alternativa que su empresa estaba esperando

Linux está en boca de todos: potencia, versatilidad, seguridad con mayúsculas, economía. Y todos se hacen las mismas preguntas. ¿Debo cambiar a **Linux**? ¿Es cierto que **Linux** es más fiable en la empresa que Windows NT, 98 o Windows 2000? ¿Es cierto que **Linux** puede convivir con Windows intercambiando información de forma transparente? ¿Puedo ahorrar con soluciones basadas en **Linux** hasta el 60% del precio en sus programas equivalentes de Microsoft? ¿Por qué más del 65% de los servidores de Internet corren bajo **Linux**? ¿Quién me dará el soporte y la garantía necesarios?

La respuesta a todas las preguntas se llama ***Universe Linux***

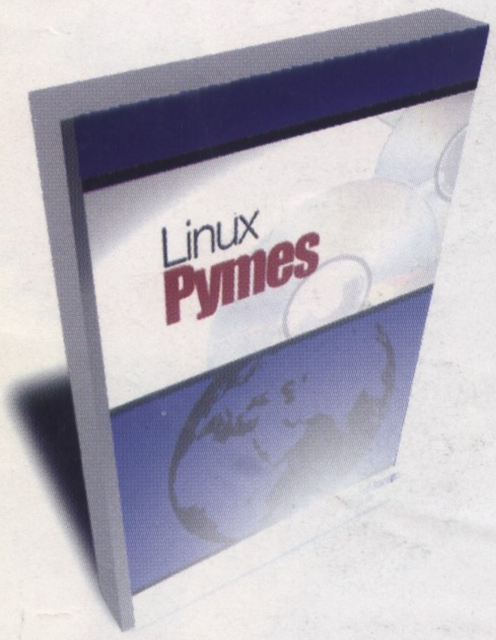
Universe Linux es la primera empresa de nuestro país dedicada exclusivamente a ofrecer soluciones basadas en **Linux** para usuarios y empresas: distribuciones **Linux**, utilidades, consultoría, instalaciones a medida, mantenimiento, formación... Todo lo que Vd. necesite y al mejor precio del mercado.

Puede que aún no sepa lo que ***Universe Linux*** puede hacer por Vd. Le invitamos a conocer nuestros productos con una demostración gratuita en nuestra sucursal de Madrid. Llámenos para concertar la demostración. Infórmese sin compromiso llamando al **91-356 69 08** o visitando nuestra página web:

http://www.u-linux.com

Llame hoy mismo. *Por fin* en LINUX hay alguien que RESPONDE.

Linux PYMES
La distribución pensada para empresas
Pvp. recomendado 9.900 pts.



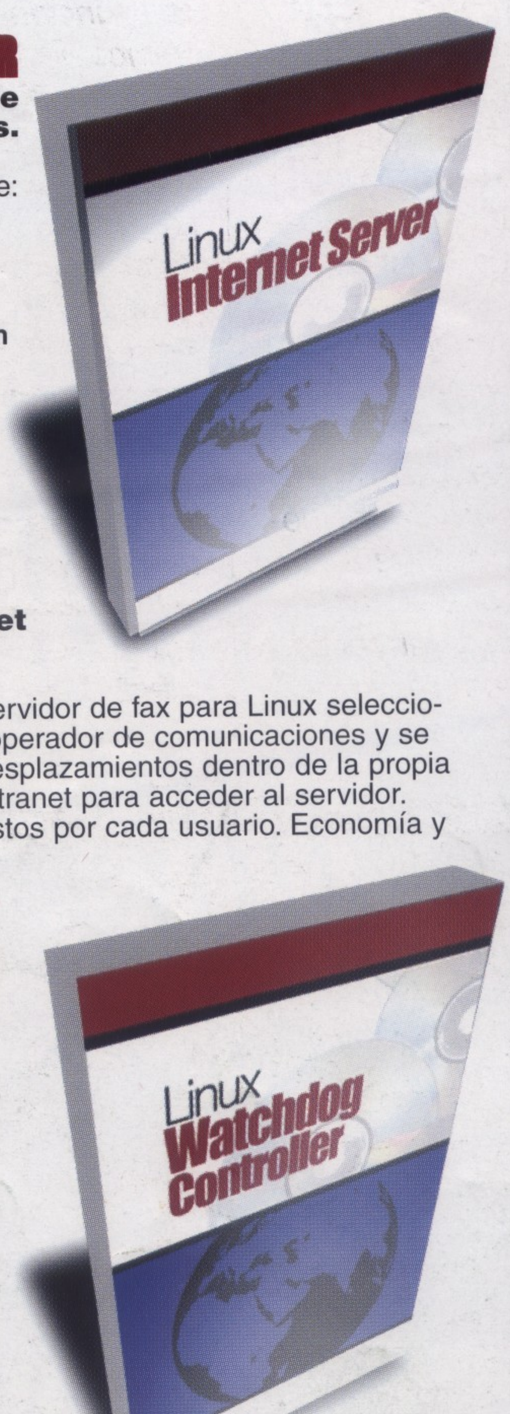
Por fin, la primera distribución creada para las PYMES: robusta, fiable, sencilla de instalar. Incluye: StarOffice, escritorio KDE, manual de instalación paso a paso y, como oferta de lanzamiento,GRATIS el programa LINUX FAX SERVER, valorado en 14.900 pts. (*)

Características:
- Instalación en modo gráfico con pantallas de ayuda
- Configuración X completa antes de la instalación de paquetes
- Flexibilidad en los modos de instalación Server y Workstation
- Utilidades de automontaje
- XFree 3.3.5-3, Kernel 2.2.12-20, KDE 1.1.2, GTK, etc.

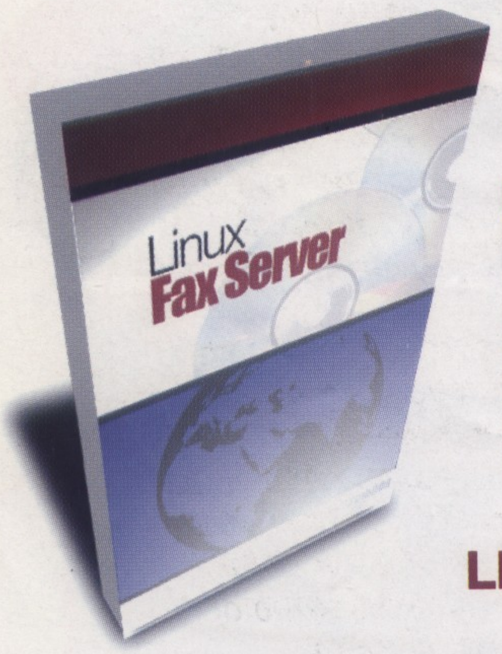
(*) Oferta válida hasta el 1 de abril del 2000.

FORMACIÓN
Cursos de Linux todos los niveles,
Básico
Administrativo
Seminarios

Linux INTERNET SERVER
El servidor más fiable
Pvp. recomendado 49.950 pts.



El servidor de Internet pensado para su empresa incluye:
***Servidor de correo electrónico, páginas Web y FTP.**
***Servidor de proxy y caché.**
***Gestión y mantenimiento de listas de correo.**
***Alojamiento del Dominio y Dirección IP fija.**
***Gestión de usuarios y autentificación de los mismos en la entrada al sistema.**
***Control y visualización de usuarios conectados.**
***10 niveles de seguridad en el servidor de páginas Web local.**
***Programación de eventos de conexión con Internet.**
Ver demostración en http:// www.u-linux.com/demo



Linux FAX SERVER
Ahorre enviando todos sus fax por la intranet
Pvp. recomendado 24.900 pts.

Se acabaron las colas para enviar un fax. El primer servidor de fax para Linux selecciona entre los tramos horarios más económicos de su operador de comunicaciones y se adapta a ellos, economizando en cada envío. Evita desplazamientos dentro de la propia empresa ahorrando tiempo y molestias al utilizar la intranet para acceder al servidor. Crea las estadísticas necesarias para analizar los gastos por cada usuario. Economía y comodidad garantizadas en sus comunicaciones.

LINUX WATCHDOG CONTROLLER
El guardián de su empresa en Internet
Pvp. recomendado 69.950 pts. (*)

El primer sistema de CONTROL TOTAL sobre Internet en su empresa. Limitación en el tiempo de conexión, listas de direcciones, chat, en general los puertos más representativos del sistema. Estadísticas de conexión por usuario, por sites accedidas, por distribución del tiempo según operador. Limitaciones individuales o por grupos de usuarios. El sistema corta la conexión cuando detecta que el usuario ha sobrepasado los límites de sus privilegios y le envía un mensaje al supervisor. LINUX WATCHDOG CONTROLLER es una herramienta imprescindible en la empresa moderna para el control del uso de Internet. El 90% de los empleados de las empresas con acceso a Internet desde su puesto de trabajo navegan con fines personales: chats, compras, etc. Esta herramienta erradica el problema realizando auditorías individualizadas de cada usuario, ya que almacena direcciones accedidas, tiempo de estancia, información recibida y enviada, etc. LINUX WATCHDOG CONTROLLER es una utilidad imprescindible para los Administradores del Sistema y Jefes de Personal. Pruébelo sin compromiso.

(*) Hasta 10 puestos de trabajo. Consultar según necesidades superiores.

Los precios no incluyen IVA.

Universe Linux *le brinda las mejores opciones para* **hacer negocio:**

Linux Training Center
Formación a todos los niveles

Si dispone de una academia de informática Vd. puede tener el LINUX TRAINING CENTER oficial de su población.
Universe Linux formará a su personal y proporcionará todo el material didáctico necesario y la certificación LINUX TRAINING CENTER. Además, le incluiremos en nuestra Lista de Centros Autorizados en todas las acciones publicitarias promocionando su negocio. (*)
La más amplia gama de cursos para todos los niveles y necesidades en formación: Introducción a Linux, Administración de Sistemas, Instalación configuración de Servidores web, Ofimática en Linux, etc.
Sistema de franquicias con número limitado de licencias por provincia.

(*) Más de 250.000 impactos cada mes en prensa especializada y general.

Linux Solution Provider
Una apuesta segura para emprendedores. Servicios en Linux

Si es Vd. socio o propietario de una empresa de Servicios Informáticos puede obtener para su provincia la representación en *exclusiva* de los productos **Universe Linux** y la certificación **LINUX SOLUTION PROVIDER** para su empresa de servicios, distribuyendo, instalando y dando soporte a la más amplia gama de productos Linux, con los mejores márgenes comerciales del mercado. Además, contará con el asesoramiento de nuestros técnicos y ayuda on line.
Un negocio de rentabilidad asegurada, sin cánones de entrada, compatible con su actividad y clientes habituales. No deje pasar la oportunidad e infórmese en el **91-356 69 08.**

Linux Developer
La mejor opción para desarrolladores

Universe Linux ofrece a los profesionales independientes del mundo de la programación la posibilidad de adquirir todos los conocimientos para trabajar en el mundo Linux. Incluye el material didáctico y el software necesario a un precio excepcional. Sólo para desarrolladores.

Y también: STAROFFICE, APPLIXWARE, ANTIVIRUS, bases de datos relacionales, servidores de ficheros, y mucho más.

Distribuidores
Universe Linux S.L. España ofrece las mejores condiciones para distribuidores, con descuentos desde el 25% hasta el 40% según tramos de compra. Infórmese sin compromiso en el teléfono:
(91) 356 69 08.
Si desea solicitar una demostración gratuita de alguno de los programas, póngase en contacto con nosotros.

Universe Linux S.L. • C/ Conde de Vilches 21 • 28028 MADRID • Teléfono: (91) 356 69 08 • Fax: (91) 355 28 95 • ***http://www.u-linux.com***

Vuestro trabajo

Los juegos de nuestros lectores

Parece que la idea de publicar gran parte de los juegos que nos habéis mandado para el concurso de Divmanía ha tenido una estupenda acogida por parte de vosotros. Muchos nos han escrito dándonos las gracias por sacar su juego y así poder enseñárselo a los "colegas". Os prometemos que, cuando nos hagamos con otro "fondo" parecido, haremos lo mismo.

Pero suponemos que no todos habréis podido disfrutar de todos los títulos que ofrecíamos el mes pasado, había más de cincuenta. Algunos no os habrán funcionado por diversos problemas con el programa o con vuestras máquinas. Aquí os transcribimos un ruego encarecido de Ferminho, un colaborador habitual de esta revista, que se lamenta por no haber podido jugar con vuestros títulos por la falta de RAM de su ordenador, y es que los tiempos adelantan que es una barbaridad:

Hola a todos. Desde hace un par de Divmanías he venido sufriendo un problema con los juegos ganadores del concurso, que no es otro que el requerimiento de memoria RAM. Mi equipo sólo tiene 16 Mb de RAM, puede que esté anticuado pero sé que habrá muchos con mi mismo problema.

Los juegos ganadores requieren de gran cantidad de memoria RAM y, de hecho, en la ultima, ninguno me funcionó. Pero me parece una pena porque cuando vi el código fuente observe que podrían haberse optimizado, solo un poquito, salvo alguna excepción, para conseguir que funcionara en un ordenador con menos de 16 megas.

El problema es que los juegos cada vez usan ficheros más grandes, con más gráficos, más sonidos, músicas, etc., y al parecer, los programas cargan cosas innecesarias en algunos momentos, y claro, eso necesita usar gran cantidad de memoria de una vez. Sin embargo, si cargárais sólo lo necesario en cada momento, esto no sucedería. Sé que la mayoría de vosotros que mandáis juegos al concurso tendréis un PC muy potente, pero por favor, pensad en nosotros los de ordenador "pobre" ^_^ gracias, y un saludo a todos. Ferminho.

Bueno, Ferminho tiene razón en todo: que el código de los juegos puede optimizarse en la mayoría de los casos y en que tiene un ordenador que, seguro, tiene merecida de sobra la jubilación.

Y ahora vamos con una ración de elogios, que no somos de piedra:

Estimados responsables de Divmanía:

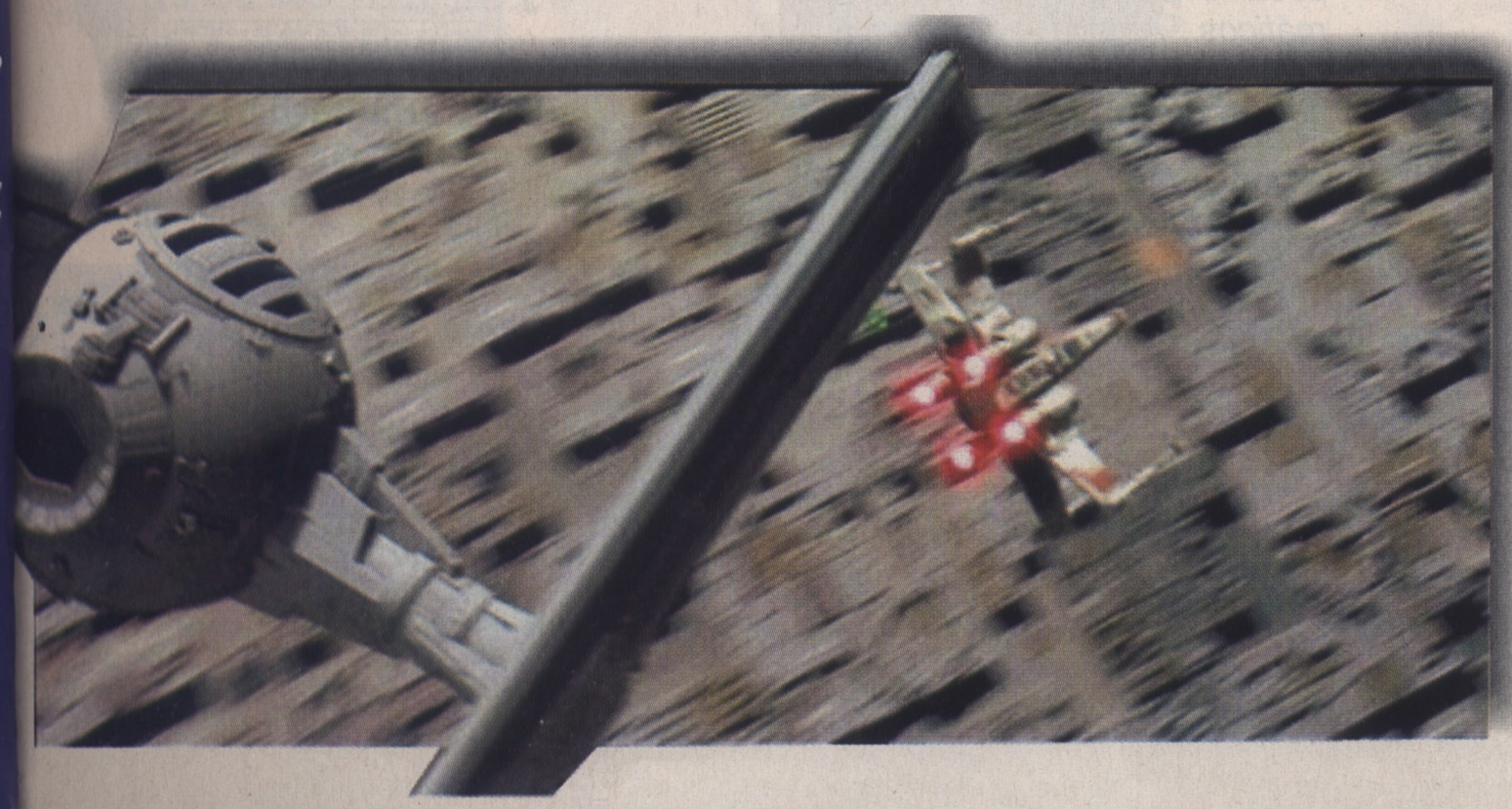
Agradecemos que, una vez más, publiquéis vuestra revista, bajo mi punto de vista la revista cada vez se orienta más hacia temas específicos y alcanza un mayor nivel, además de ser un punto de partida y de guía para todos aquellos programadores que deciden caminar en el mundo de los videojuegos.

Queremos hacer llegar este mail con el fin de agradecer a todas las personas que hacen posible Divmanía, y en especial a Alfredo del Barrio los comentarios que se realizaron en Divmanía número 5 en la sección WEB DIV sobre el grupo de programación Dark Illusions.

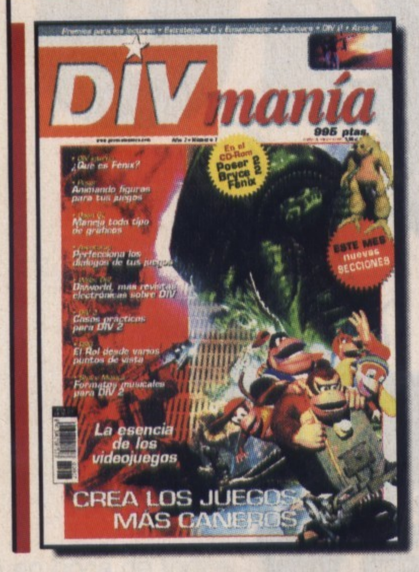
Os sugerimos que sigáis con ésta línea de trabajo.

Un saludo: The Machine (coordinador de Dark Illusions)

Gracias por vuestro apoyo. De regalo este grupo de desarrollo nos ha mandado un adictivo juegecito, el conocido Master Mind, con el logo de la revista. Para que todos lo disfrutéis lo hemos metido en el CD-Rom que acompaña a la revista. Hasta el próximo número.



DIVmanía



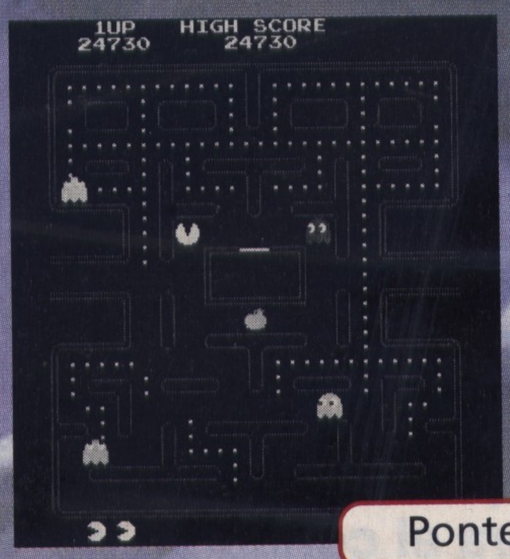
¿Amas la programación de juegos...? Esta es tu revista

Un saludo

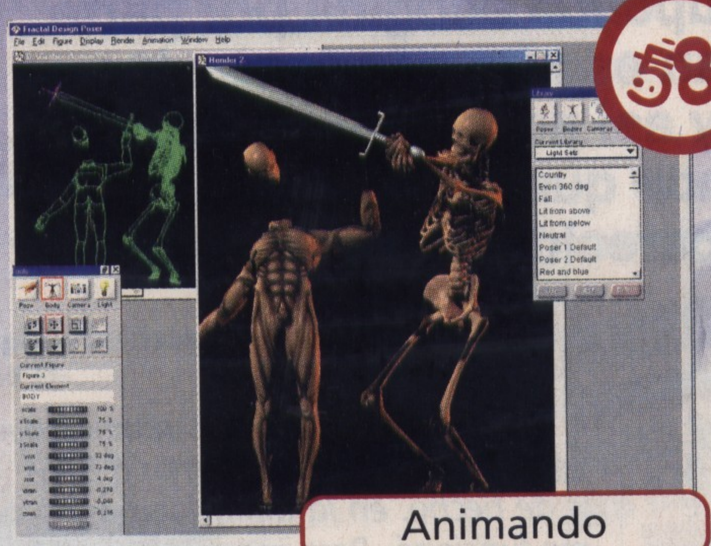
Un número más estamos en el kiosco. Por el momento no sólo hemos conseguido ser fieles a nuestra cita cada dos meses sino que cada vez damos más que hablar. Como ejemplo basta un botón: al pie de estas líneas podéis ver la imagen de un cartel que se encontraba pegado en una parada de autobús. Como podéis ver se trata de un anuncio de una academia de informática (a la que no vamos a nombrar). Pues bien, si os fijáis bien veréis el nombre de un programa bien conocido por todos nosotros. Efectivamente: DIV. Y es que nuestro pequeño programa se hace un hueco entre los grandes.

Por lo demás, en este número hemos ampliado nuestras miras. Nuevas secciones como Poser y Open GL, e información de primera mano sobre un programa del que seguro habréis oído hablar, Fénix, vienen a unirse a la revista a la espera de que sean muy útiles para vosotros. De hecho, la idea de estas secciones ha surgido de vosotros. De igual modo hemos tratado de recoger en el Cd las peticiones que nos habéis hecho. Muchas gracias a todos por la colaboración porque vosotros sois los principales culpables de que esta revista siga adelante.

Año 2 · Número 7



Ponte al día



Animando

14 REPORTAJE

Subiendo al Podium

Un reportaje que trata de desentrañar la esencia de los videojuegos. Solamente cuando comprendamos qué se necesita para programar un "número 1" podremos iniciar nuestros trabajos con las ideas claras. Superarse a sí mismo, el dominio de reglas técnicas y ponerse en el lugar del personaje es lo que busca todo jugador de videojuegos.

58 ESPECIAL

Poser

Poser es una utilidad que permite animar figuras humanas, dotándolas de total libertad de movimientos, renderizarlas, y visualizarlas desde cualquier punto de vista. Incluye muchas más figuras que el Generador de Sprites de DIV 2, las figuras son totalmente articulables y configurables. Incluso expresiones podremos crear con este programa.

DIV Developer

2 CURSO DE PROGRAMACIÓN BASICA

Para los que empiezan

Programar es una tarea difícil. Menos mal que nuestro curso de programación básica está aquí para facilitarte las cosas

6 CURSO DE ENSAMBLADOR

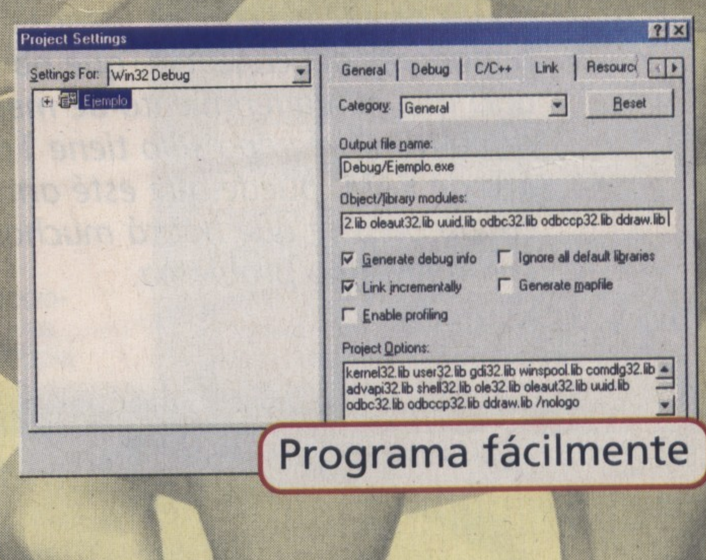
Imprescindible

Todo lo que debes saber para convertir las instrucciones escritas en texto a bytes. No habrá formato binario que pueda pararte.

4 PROGRAMACIÓN EN C

Un clásico en la programación

Nuevo capítulo de uno de los lenguajes más utilizados en el desarrollo de programas informáticos. Amplia tus conocimientos para que no haya código que se te resista.



Programa fácilmente

Director: Mario Luis
mluis@prensatecnica.com

Coordinador Técnico: Oscar Condés
gover@prensatecnica.com

Colaboradores: Pablo Trinidad, Sergio Cánovas, Miguel Barroso, David Martínez, Emilio Llamas, Javier Fernández, Fermín Vicente, Ramón de España, Santiago Orgaz, Antonio Marchal, Santiago García y Jordi Martín.

Edición: Alfredo del Barrio y Guillermo Izquierdo.

Dirección de Arte: Francisco Calero

Jefe de Departamento: José A. Gil

Maquetación: Antonio García Tomé, Antonio Barbero, Ana Isabel Madero, Jesús Mena, Oscar Menoyo y Diana Sarmiento

Portada: Francisco A. Anguís

Publicidad: Marisa Fernández,
marisa@prensatecnica.com
Sonia Glez.-Villamil, Noelia Menéndez y Emerio Arena

Supervisión CD-Rom:
Alvaro García

Servicio Técnico CD-Rom:
Eugenio García
Horario de atención:
tardes 16:00 - 18:00 h
E-mail: *stecnico@prensatecnica.com*

Secretaria de Redacción:
Elena Fernández

Departamento de Suscripciones:
Sandra Fernández y Noemí Iscart
suscripciones@prensatecnica.com

Departamento de Administración:
Juan López, Juan Ignacio Domínguez

Departamento Comercial:
Marcelino Ormeño

REDACCIÓN, PUBLICIDAD Y ADMINISTRACIÓN
c/ Alfonso Gómez 42. Nave 1.1.2
Madrid 28037. España
Tfno: 91. 304. 06. 22
Fax: 91. 304. 17. 97
Si llama desde fuera de España, marcar (+34)

Sumario

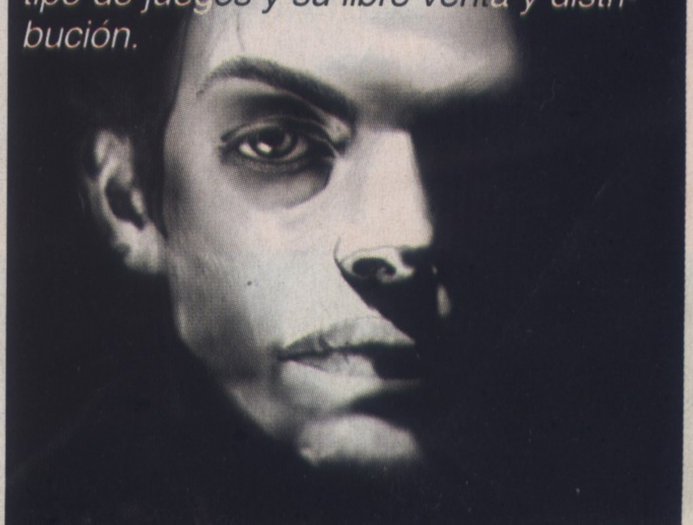
Programas del lector

PREMIAMOS LOS MEJORES JUEGOS DE LOS LECTORES

DIV ha creado toda una escuela dentro del mundo de la programación. Todos los que se han acercado a la herramienta han sido capaces de programar. Por ello realizamos un concurso entre los lectores que hayan realizado juegos con DIV. Os ofrecemos un primer premio de 25.000 pesetas y dos accésit de 20.000 pesetas.

¿QUÉ ES DIV GAMES STUDIO?

DIV Games Studio es una herramienta de programación que facilita en gran manera nuestra inmersión en el software de entretenimiento. Es el primer entorno profesional que permite realizar videojuegos con fines comerciales sin necesidad de un pago adicional. Con el carné de desarrollador incluido se permite el desarrollo de cualquier tipo de juegos y su libre venta y distribución.



E-mail: gover@prensatecnica.com
http://www.prensatecnica.com
Horario de atención al público:
de 09:00 a 19:00 h
ininterrumpidamente
EDITA: PRENSA TÉCNICA
Director General: Mario Luis
Director Editorial: Eduardo Toribio
Director Técnico: Fernando Escudero
Director de Producción: C.P. Cerezo
Directora Publicidad: Marisa Fernández
Director Comercial: Esteban Martínez
Fotomecánica: Prensa Técnica
Impresión: I.G.Printone, S.A.
Duplicación del CD-Rom: M.P. O., Servicios Ibéricos, Grupo Cóndor
Distribución: SGEL. Avda Valdelaparra, 29 Alcobendas. Madrid

Depósito legal: M-42077-1998
AÑO 2 • NÚMERO 7

Las Tecnologías más avanzadas de la web a su alcanc

onviér
n espe
de

java

El lenguaje universal para aplicaciones Web

cgi

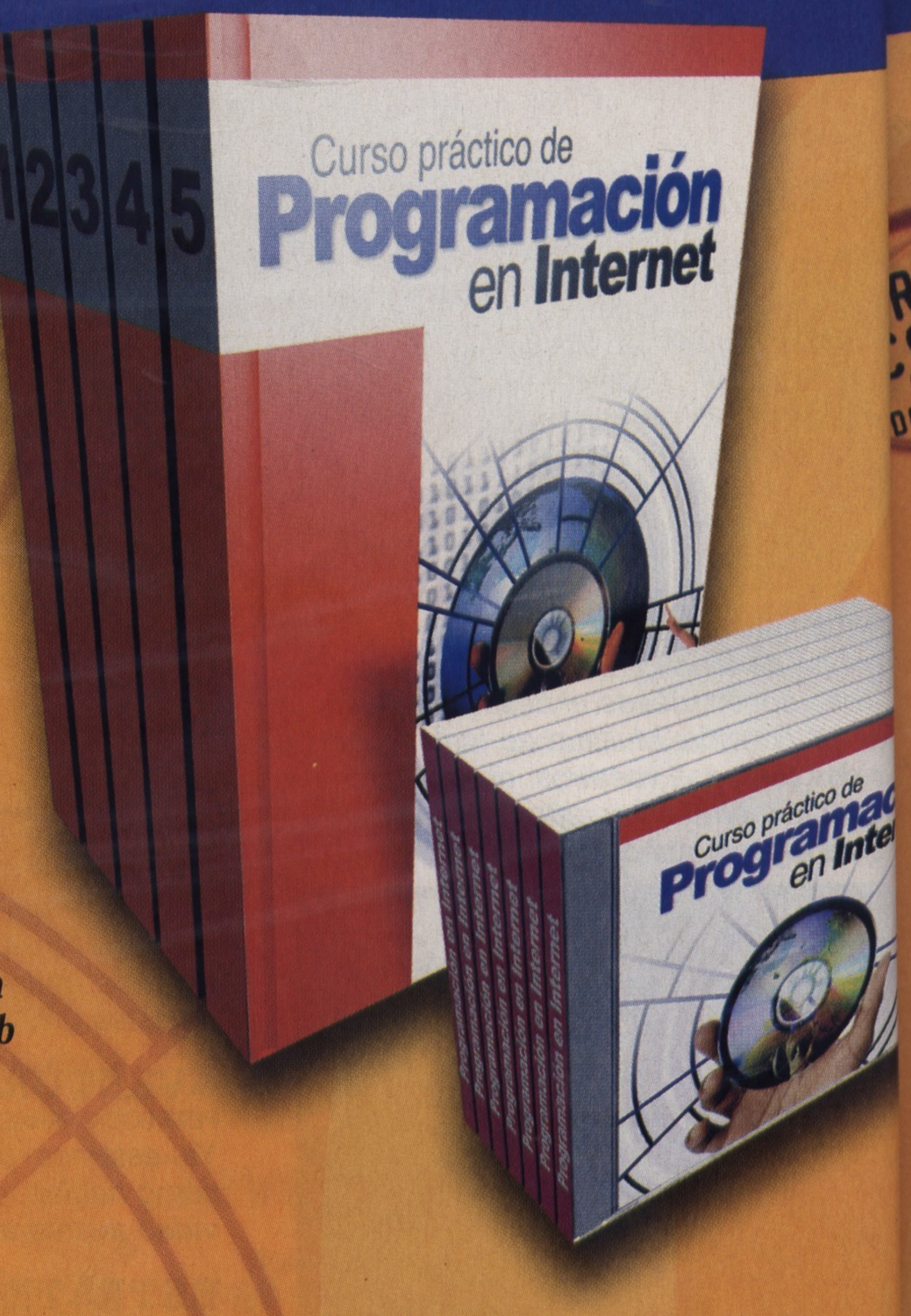
Interfaz para desarrollo de aplicaciones en Servidores Web

html

Lenguaje de creación de páginas Web

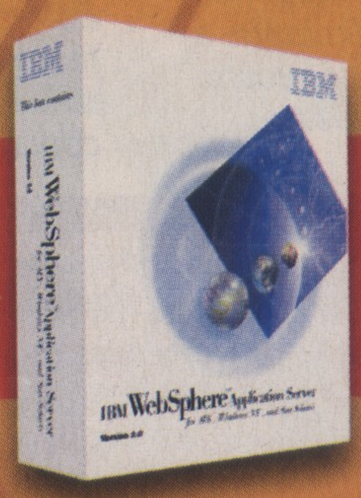
java script

Lenguaje de scripts para clientes y servidores Web



Curso prác d

Prog a



Sorteamos

5 WebSphere Application Server
Avanzado para NT, AIX y Solaris

Cada uno de ellos valorado en 1.000.000 pt

onviértase en
n especialista
de la Web

PROGRAMAS COMPLETOS
PARA PRACTICAR
DESDE EL PRIMER DÍA

Ya a la venta en tu quiosco
los números 1 y 2 con 3 CD-Roms
por sólo

995 ptas.

5,98 Euros

Introducción
Curso práctico de
Programación en Internet
Java • CGI • HTML • JavaScript
1

Teoría
• Estructura de la obra

Práctica
• Primeros pasos en HTML

Tutorial
• JBuilder y Visual Java ++

CD-Rom
• Visual Café 2.0

995 ptas. 0,00 €

8 413042 087298 00009

Prens@ Técnic de libros y publicaciones

Primeros pasos
Curso práctico de
Programación en Internet
Java • CGI • HTML • JavaScript
2

- *La mejor forma de entrar de lleno en el mundo de Internet: todo sobre las comunicaciones en Internet y el flujo de datos en el World Wide Web.*

- *Todo lo necesario para desarrollar con profesionalidad aplicaciones para Internet en un curso de cinco tomos y 50 fascículos acompañados de CD-Roms en los que encontrar las herramientas más útiles para el desarrollador web*

Infórmate en el teléfono: 91 304 06 22

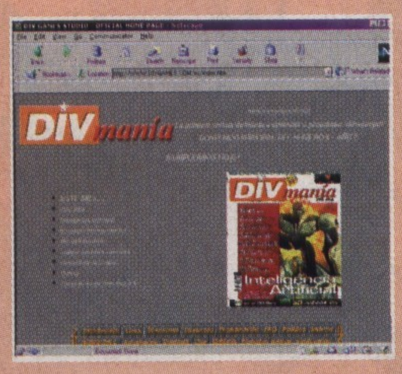
Prens@ Técnic de libros y publicaciones

Edita PRENSA TÉCNICA
Alfonso Gómez, 42 - Nave 1-1-2
28037 MADRID
Teléfono: 91 304 06 22 - Fax: 91 304 17 97
http://www.prensatecnica.com

Nuevo webmaster para la página oficial de DIV

El antiguo webmaster de www.divgames.com ha delegado su cargo en Roman1, un usuario frecuente por el canal, que se encargará a partir de ahora del contenido de la web oficial.

En ella podréis encontrar parte del contenido de esta revista en la sección DIVmanía. O alguna noticia bastante interesante como la que dice "En una interesantísima rueda de prensa Daniel Navarro, programador principal de DIV, anunció que tiene la intención de lanzar una nueva versión, denominada DIV Profesional, a mediados de este mismo año".



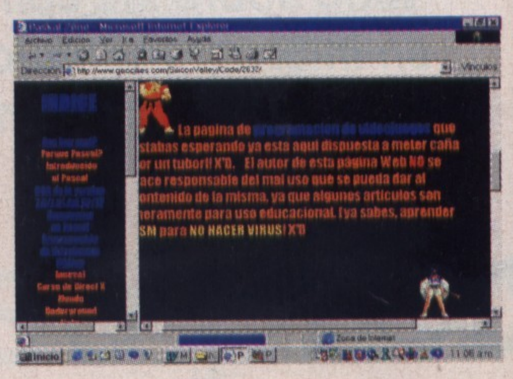
Noticias sobre videojuegos

Otra página web: www.dreamers.com realizado por Kokomos Studio puede ser otra visita interesante. Kokomos Studio es un estudio creativo formado por profesionales de diversas especialidades como la animación, el cómic, los videojuegos, Internet, edición y diseño, etc. Su objetivo es mantener una web en la que lo que prime sea la información actual y la inmediatez. Se ofrecen noticiarios especializados sobre temas diversos (cine y TV, cómic, etc). Además encontraréis noticias de última hora sobre cualquier tema relacionado con los videojuegos, tanto para PC como para cualquier plataforma, así como enlaces de ultimísima actualidad. En esta página está disponible el juego *Drakoon*, programado por Miguel Chaves. Es una aventura gráfica creada con Javascrip y DHTML y que quizá os sea de utilidad para vuestros trabajos o simplemente como mera diversión.

Páginas de programación de videojuegos

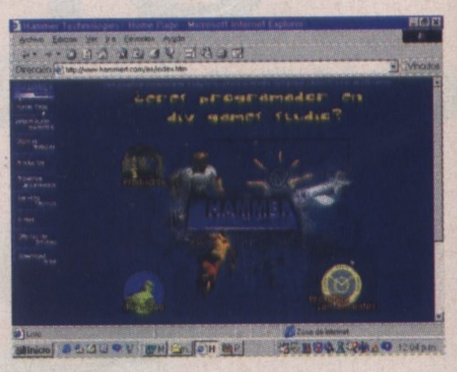
Gracias a un lector, llamado José, podemos informaros de la dirección de una página web dedicada totalmente al desarrollo y programación de videojuegos. La dirección es la siguiente: http://www.geocities.com/SiliconValley/Code/2632/

Será próximamente analizada en la sección "Web div" de esta revista, pero a modo de entremés, deciros que esta página está totalmente en castellano y posee extensa información para todos aquellos que se inician en el mundo de la creación de videojuegos. Es bastante útil porque combina las explicaciones detalladas de cómo realizar una tarea determinada, además del código fuente correspondiente. La web explica desde como dibujar sprites, hacer transparencias (efecto de luces dinámicas), rutinas para controlar disparos, interacción con los decorados, técnicas de scroll parallax y de scrolles por tiles, y un largisimo etcetera. Como ejemplo se incluye el código fuente del Kutre Mario Kros, como muestra de lo que se puede llegar a hacer con el material que hay en la web. Gracias José.



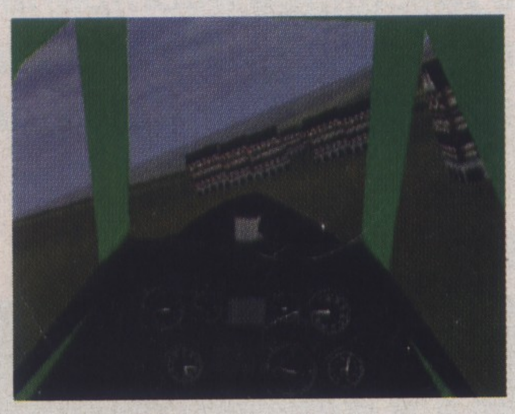
Actualización de la página de Hammer

Hammer Technologies dispone de una página web que siempre es interesante visitar, y en la que se acaba de producir una actualización de contenidos. Lo más destacable es la presentación de las próximas novedades de la compañía en la sección "Próximos lanzamientos", donde se ofrece información sobre los dos videojuegos que se estrenarán en los próximos meses (*"Hellfire. In the heat of war"* y *"Jagdverband 44"*). El primero de ellos es un simulador de helicópteros de combate con elementos estilo arcade.



El apartado de "Productos" recoge información sobre varios juegos ya editados, como *"Tie Break Tennis 98"*, *"Toby"*, *"Ancient Evil"* o *"Mad Trax"*, así como sobre el programa para la creación de videojuegos "DIV-2 Games Studio"

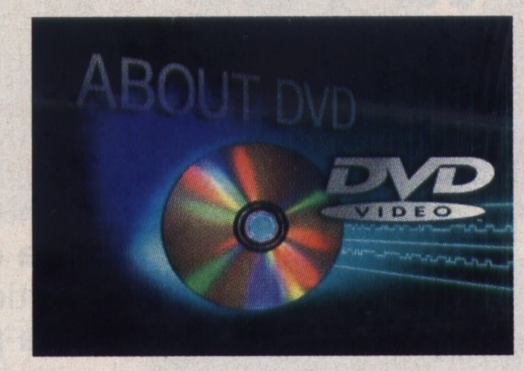
También se informa sobre los requisitos para acceder al departamento de creación de videojuegos. "Hammer Technologies" pretende descubrir nuevos talentos en este campo que ayuden a esta desarrolladora española a seguir creciendo con igual o mayor determinación que hasta ahora.



El código del DVD se sigue distribuyendo

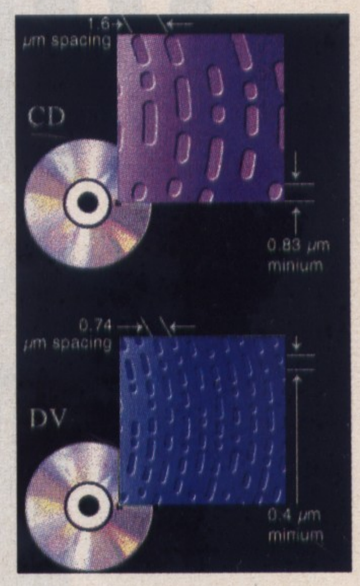
Los hackers de todo el mundo parece que están ganando algunas pequeñas escaramuzas judiciales contra las grandes compañías multinacionales del sector de la electrónica. Hace poco, el juez William Efving tomo la decisión de no bloquear la distribución a través de Internet del código hackeado que puede desencriptar la información contenida en un DVD, conseguida por un grupo de hackers noruego y prontamente distribuido a través de Internet.

La agencia DVD, Copy Control Association Inc., había solicitado al susodicho juez que decenas de páginas web retirasen el programa que permite desencriptar los DVD. La solicitud fue denegada por el juez.



Como todos sabréis, el DVD era el único formato que se había salvado de la piratería, pero ya, su guardado código de encriptación, es un secreto a voces.

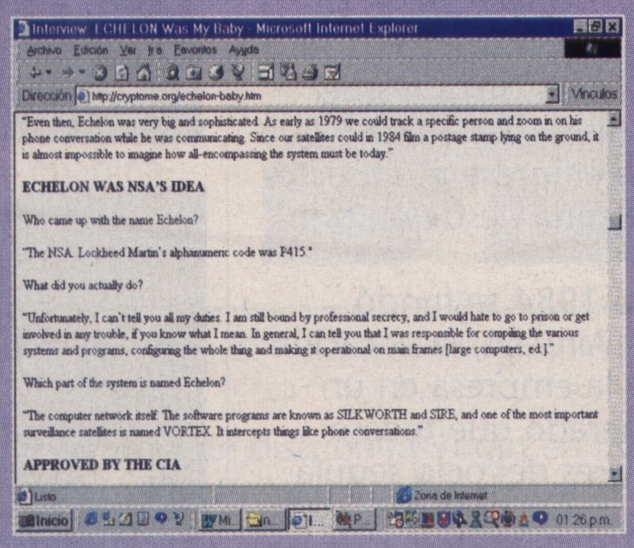
Nosotros tenemos una teoría propia y algo arriesgada: ¿puede ser que las propias empresas creadoras del DVD hayan hecho público el código para fomentar una tecnología que, hasta hace poco, había tenido poca penetración en el mercado, entre otras causas por la imposibilidad de piratear los DVD? Nosotros no lo sabemos, si tenéis alguna idea hacédnosla saber.



Echelon, el control invisible

Margaret Newsham, ex programadora de la red fantasma de espionaje mundial electrónico, conocida como Echelon, declaró hace poco a un periódico danés lo siguiente: "Si os lo pudiese contar todo, entenderíais que Echelon es muy grande, tanto que desafía la comprensión humana". Margaret vive atemorizada por las posibles represalias que pueda sufrir por parte de sus antiguos jefes.

Echelon es el nombre de la red de ordenadores que utiliza la NSA (Agencia de Seguridad Nacional) de EEUU para espiar correo electrónico, teléfono y fax mediante satélites y ordenadores.


Interview: ECHELON Was My Baby - Microsoft Internet Explorer

Archivo Edición Ver Ir a Favoritos Ayuda

Dirección http://cryptome.org/echelon-baby.htm Vínculos

"Even then, Echelon was very big and sophisticated. As early as 1979 we could track a specific person and zoom in on his phone conversation while he was communicating. Since our satellites could in 1984 film a postage stamp lying on the ground, it is almost impossible to imagine how all-encompassing the system must be today."

ECHELON WAS NSA'S IDEA

Who came up with the name Echelon?

"The NSA. Lockheed Martin's alphanumeric code was P415."

What did you actually do?

"Unfortunately, I can't tell you all my duties. I am still bound by professional secrecy, and I would hate to go to prison or get involved in any trouble, if you know what I mean. In general, I can tell you that I was responsible for compiling the various systems and programs, configuring the whole thing and making it operational on main frames [large computers, ed.]."

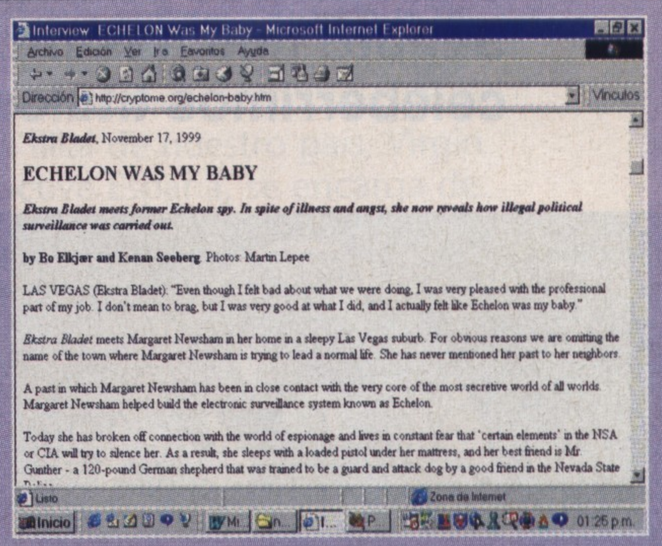
Which part of the system is named Echelon?

"The computer network itself. The software programs are known as SILKWORTH and SIRE, and one of the most important surveillance satellites is named VORTEX. It intercepts things like phone conversations."

APPROVED BY THE CIA

Listo Zona de Internet

Inicio 01:26 p.m.


Interview: ECHELON Was My Baby - Microsoft Internet Explorer

Archivo Edición Ver Ir a Favoritos Ayuda

Dirección http://cryptome.org/echelon-baby.htm Vínculos

Ekstra Bladet, November 17, 1999

ECHELON WAS MY BABY

Ekstra Bladet meets former Echelon spy. In spite of illness and angst, she now reveals how illegal political surveillance was carried out.

by Bo Elkjær and Kenan Seeberg. Photos: Martin Lepee

LAS VEGAS (Ekstra Bladet): "Even though I felt bad about what we were doing, I was very pleased with the professional part of my job. I don't mean to brag, but I was very good at what I did, and I actually felt like Echelon was my baby."

Ekstra Bladet meets Margaret Newsham in her home in a sleepy Las Vegas suburb. For obvious reasons we are omitting the name of the town where Margaret Newsham is trying to lead a normal life. She has never mentioned her past to her neighbors.

A past in which Margaret Newsham has been in close contact with the very core of the most secretive world of all worlds. Margaret Newsham helped build the electronic surveillance system known as Echelon.

Today she has broken off connection with the world of espionage and lives in constant fear that 'certain elements' in the NSA or CIA will try to silence her. As a result, she sleeps with a loaded pistol under her mattress, and her best friend is Mr. Gunther - a 120-pound German shepherd that was trained to be a guard and attack dog by a good friend in the Nevada State

Listo Zona de Internet

Inicio 01:25 p.m.

La ex programadora afirma que, si alguien en el mundo menciona su nombre en un correo electrónico, el mensaje será "interceptado, analizado, coordinado, reenviado y registrado". También ocurre cuando se escriben las palabras: bomba, Clinton, presidente, etc.

Si quieres que tu mail acabe no se sabe dónde, puedes hacer la prueba escribiendo alguna de estas palabras, claro que tú no te enterarás de nada. Si queréis acceder a la entrevista completa la encontraréis en: cryptome.org/echelon-baby.htm

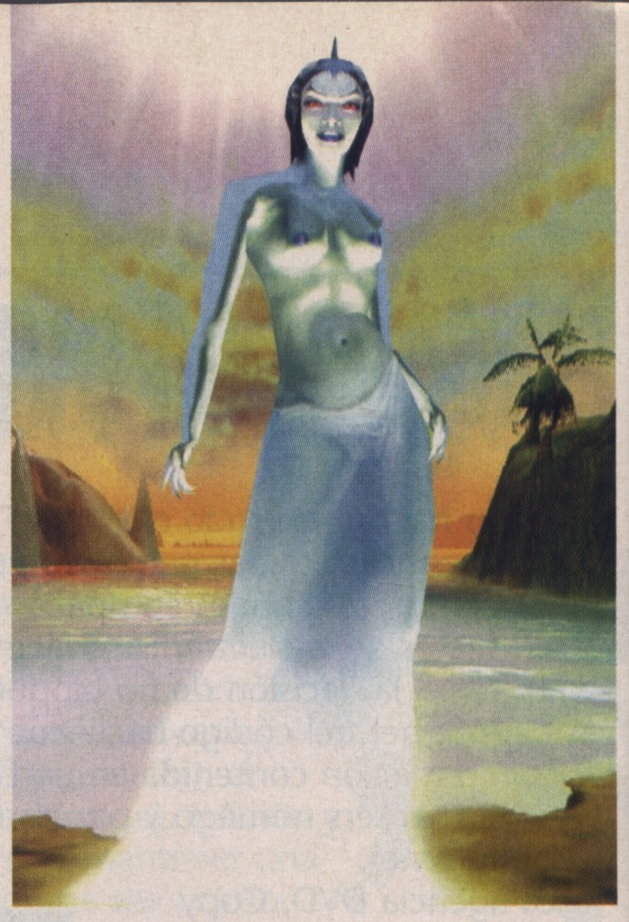
Kedadas

Los diveros de Madrid tienen una cita los últimos domingos de cada mes en las puertas de la Confederación de los Recios (Centro Mail de Callao, metro Callao) a las diez y media de la mañana.

El programa de estas reuniones es el siguiente: desde las diez y media hasta las doce se habla de diversos temas; cuando abren Centro Mail, sobre las doce, se suele jugar una hora a *Half-Life* en red (hay mucho vicio); luego toca comer en algún restaurante de comida rápida y para finalizar, los que quieren, se suelen ir a tomar un café.

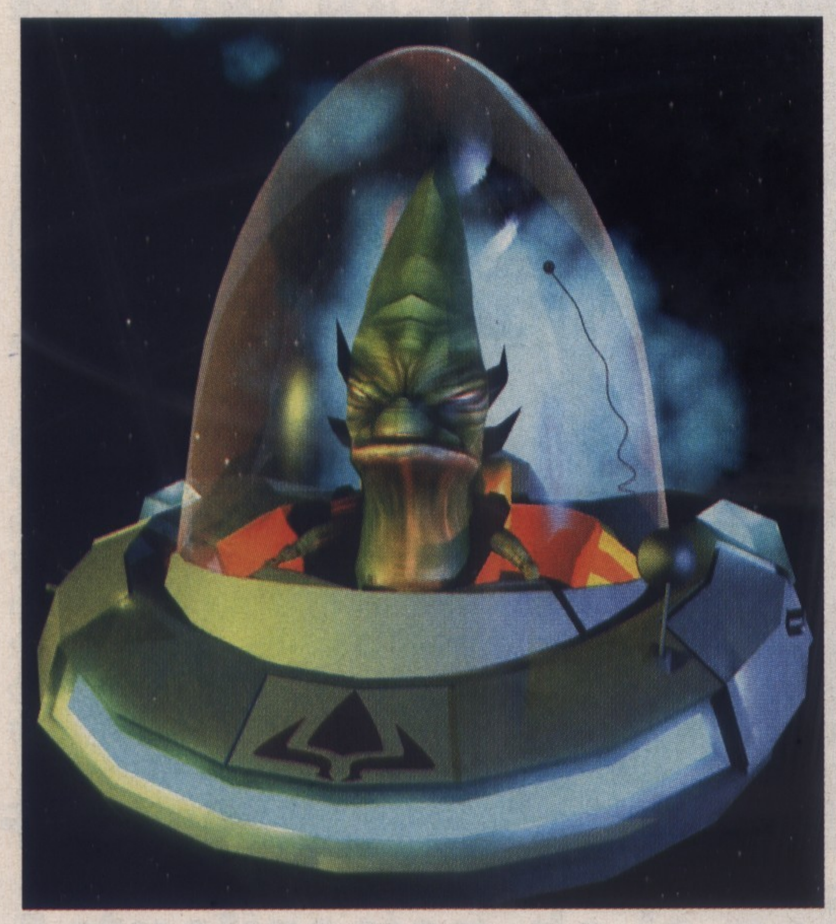
La foto que podéis ver es la de la primera kedada. De izquierda a derecha, empezando por el fondo, aparecen: Ferminho, Guy y Zyope. Los de delante son Dumbo, Manowar, Oscar BrainDead, UAMike y Skater. Todo el mundo está invitado a la próxima reunión.

Virgin



Un imperio al servicio de tu ocio

Virgin, una compañía que empezó editando música, ¿quién no recuerda alguna canción de Mike Oldfield?, tiene una división entera, Virgin Interactive, dedicada íntegramente al desarrollo, publicación y distribución de videojuegos para todo tipo de plataformas. Seguro que has probado alguno de sus celebérrimos títulos en alguna ocasión.

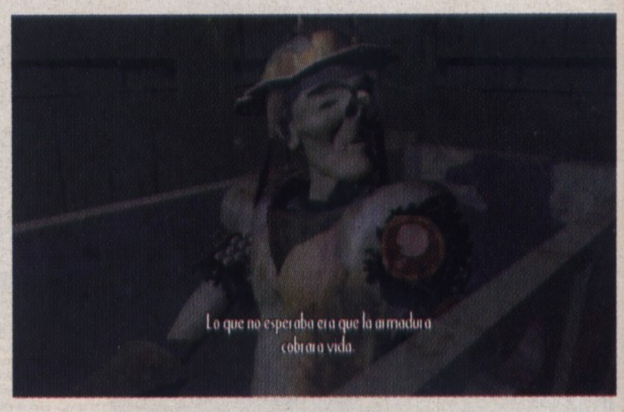


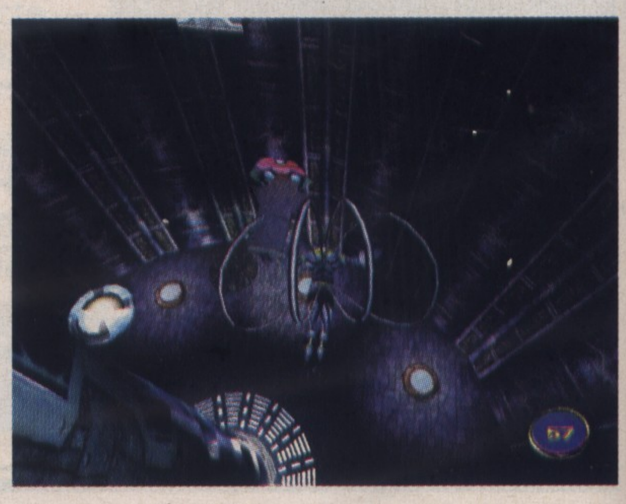
La casa Virgin es sobradamente conocida por todos los aficionados a matar el rato delante de una pantalla de ordenador o de consola viviendo aventuras virtuales. Pero Virgin empezó siendo una de las pocas compañías independientes de edición de música que se convirtió en una multinacional por sus propios medios. Vamos a ver un breve repaso de su recorrido.

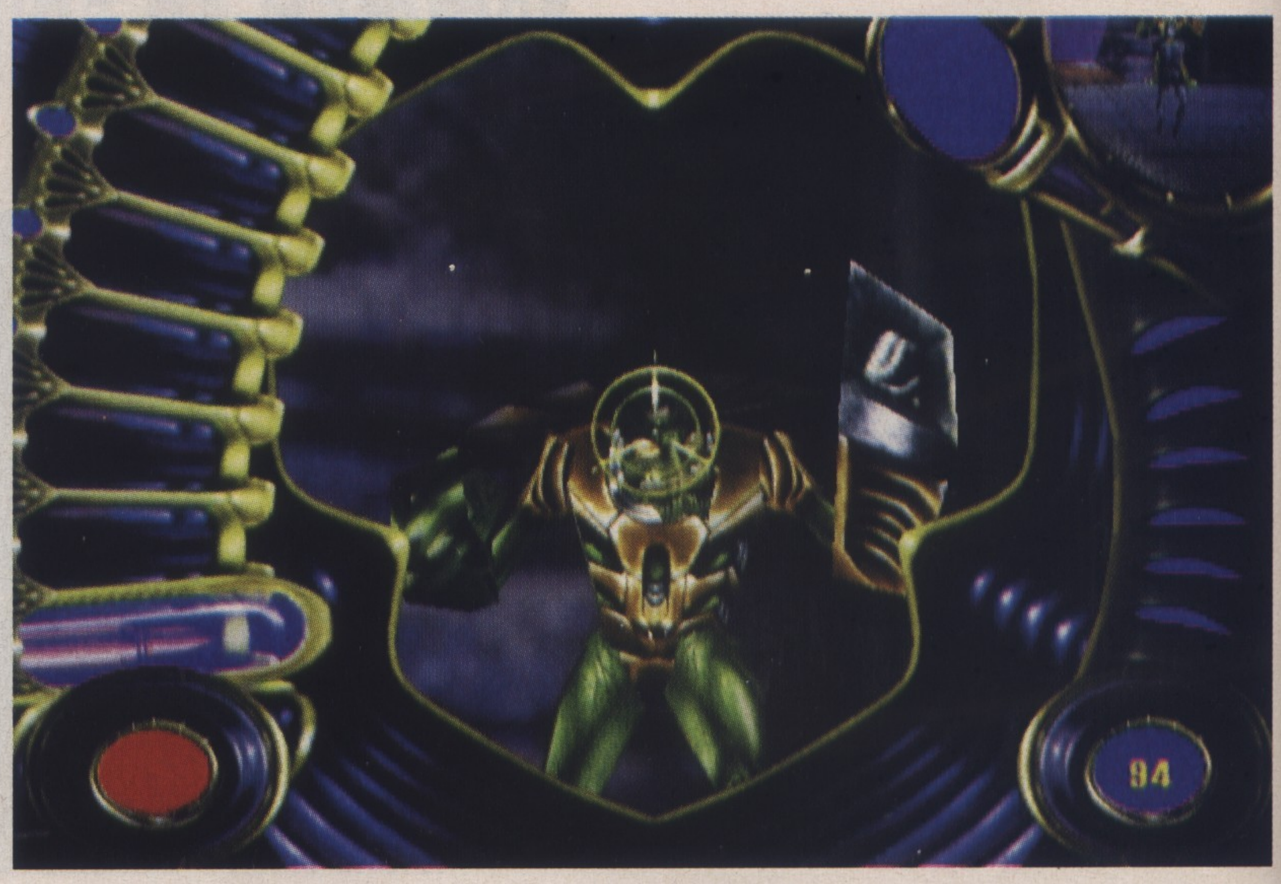
Un ascenso meteórico

Virgin fue fundada en Londres por Richard Branson y algunos amigos a finales de los años 60. Teniendo claro que lo que querían era montar una compañía de discos, les llevó más tiempo decidir que nombre poner a la nueva compañía. Después de unos días de discusiones se decidieron por bautizarla como Virgin, nombre moderno, que sonaba bien, y que, ya desde los comienzos de la empresa, se pensaba que podía ser adecuado para usarlo en otros sectores del ocio, no sólo musicales. Parece que la vocación de empresa multisectorial estaba decidida desde un principio.

Cuando en 1984 se fundó Virgin Atlantic Airways, la vocación de convertir a la empresa en un gran conglomerado que tocase todos los sectores del ocio seguía en pleno desarrollo. Pero la filosofía que ha regido la empresa, desde sus comienzos como sello independiente, ha sido la frase: "lo pequeño es hermoso". Para que este lema siguiese en plena vigencia en una compañía que se había convertido, ya a la altura de los 90, en una poderosa multinacional, se siguió una práctica que parece que ha sido bastante productiva y exitosa: cada división de Virgin es totalmente autosuficiente y autónoma a la hora de afrontar sus propios negocios. Si hubiese que bus-







car un símil político, se podría decir que Virgin ha sido un curioso ejemplo de "grupo de compañías confederadas".

Hoy por hoy, el imperio Virgin se compone de más de 200 compañías que están dispersas a todo lo largo y ancho de los cinco continentes, tiene más de 25.000 empleados, y mueve más de cinco mil millones de dólares en una gran variedad de negocios y áreas comerciales. Editoras de música y de películas, compañías de viajes, centros comerciales, empresas de bebidas, radios, televisiones, libros, hoteles, ropa, cosméticos, agencias de modelos, viajes en globo, servicios de catering; en fin todo lo que se te pueda pasar por la imaginación, incluso la propiedad de un equipo de rugby.

Virgin Interactive

Hoy por hoy, los negocios de Virgin Interactive, la filial que se dedica al desarrollo, publicación y distribución de videojuegos, van unidos a la empresa Titus que se hizo con el control de la primera recientemente.

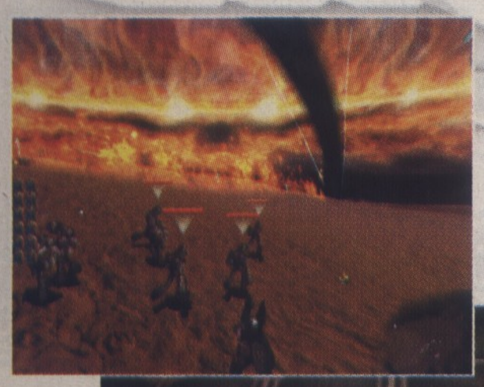
La filial de nuestro país, Virgin Interactive España, se encarga de publicar y distribuir los videojuegos de multitud de desarrolladoras de todo el mundo. Entre las más famosas están Interplay, que posee los derechos de Black Isle e Interplay, creadoras de *Baldur´s Gate* y *Planescape Torment*; GT, distribuidora de *Imperium Galactica* o *Panzer Elite*; Titus, Capcom, Microids o Cryo.

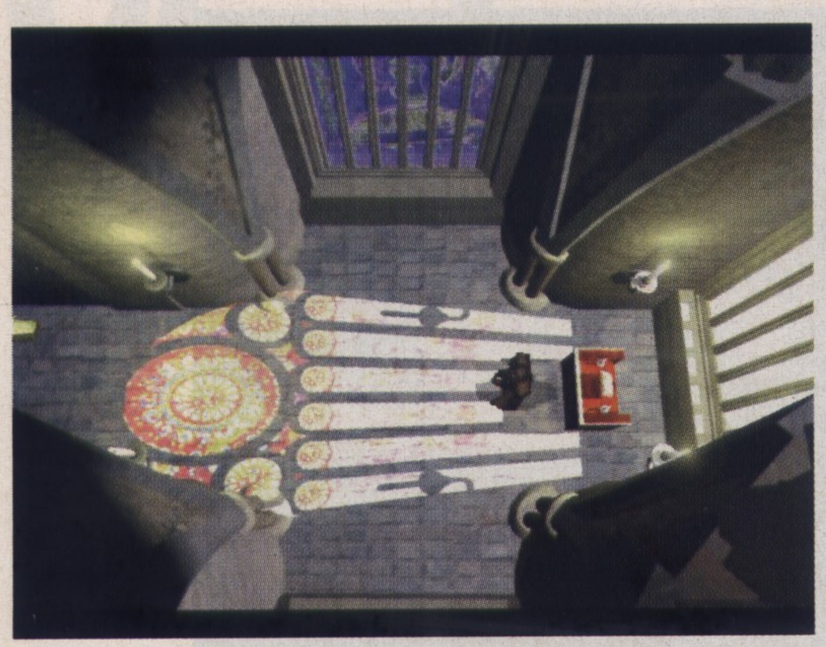
Los tentáculos del Imperio Virgin se ramifican por todos los sectores del ocio tradicional y mediático

Interplay

Los títulos que próximamente distribuirá Virgin en España, fruto de sus acuerdos con Interplay, son muchos, y todos de gran calidad. Entre los juegos que próximamente aterrizarán en nuestro país están algunos de los más esperados por los consumidores de software lúdico.

Tomad nota: *Messiah*, un juego que romperá esquemas y en el que tomaremos el papel de un angelito capaz de poseer el cuerpo de cualquiera; *MDK 2*, un arcade en tres dimensiones y perspectiva en tercera persona en el que podremos manejar a tres personajes; *Icewind Dale*, un título de rol inspirado en la franquicia Dungeons & Dragons; *Evolva*, del género estratégico; *Baldur´s Gate 2*, segunda parte de uno de los juegos de rol más aclamados de todos los tiempos; *Sacrifice, Giants* o *Star Treck New Worlds*. Seguro que a más de uno se le está haciendo la boca agua.





Todo un universo de títulos para todo tipo de plataformas que, seguro, te llevarán a otras dimensiones. Esperamos ansiosamente que los juegos de Virgin Interactive alimenten nuevamente a nuestras voraces máquinas.

Alfredo del Barrio

GT

Virgin se ha hecho recientemente con los derechos de distribución de los títulos que posee la empresa GT Interactive Software, una buena inversión creemos nosotros, pasamos a detallároslos.
Entre la lista de títulos de GT que puedes encontrar en este momento en el mercado están: *40 Winks* y *Discworld Noir* para la consola PlayStation; *Imperium Galáctica 2: Alliances*, estrategia en otros mundos; *Nations Fighter Command*, simulador de aviones ambientado en la II Guerra Mundial; *Panzer Elite*, que también nos llevará a la última gran conflagración bélica, pero esta vez a las entrañas de un tanque; *Wheel of Time* un juego de rol/arcade en tres dimensiones; o el aclamado *Unreal Tournament*, que no necesita ni presentación.

Y más

Mucho más, es lo que nos ofrece Virgin Interactive en su catálogo. Los juegos son tantos que no van a caber en estas páginas, pero por lo menos intentaremos mencionarlos aunque sea de pasada.
De Cryo se van a distribuir las versiones para PlayStation de Atlantis y Egipto, dos aventuras gráficas que nos adentrarán en ambientes misteriosos y exóticos.
De Capcom la versión de *Resident Evil 2* para la consola Nintendo 64, *Plasma Sword* o *Street Fighter Alpha 3* para la Dreamcast; *Jojo´s Bizarre Adventure* o *Street Fighter EX 2 Plus* para la PlayStation; o *Dino Crisis* para PC.
3DO proveerá a Virgin de jugosos títulos de la serie de soldaditos de plástico *Army Men. In Space, Air Attack* o *Sarge Heroes* han salido o están a punto de salir. También saldrá *Crusader of Might & Magic* o *Global Assault*.

Y paramos ya porque estamos sin resuello, no sin antes avisar que los títulos que nos hemos dejado en el tintero, o mejor en el disco duro de nuestro ordenador, son multitud.

Mega GAMES 1

Estrategia
Knights & Merchants

Acción Total
Emergency

"Megagames 1" te ofrece dos videojuegos por el precio de uno. "Emergency" y "Knights and Merchants" son dos juegos de estrategia con los que desarrollarás tu sangre fría y tu faceta de estratega militar.

DIGITAL DREAMS MULTIMEDIA
C/Alfonso Gómez, 42, nave 1-1-2
28037 Madrid (spain)
Phone: 91 304 06 22 Fax:91 304 17 97
http://www.ddmultimedia.com

Digital Dreams MULTIMEDIA

COMPATIBLE WINDOWS 98

PC CD rom **Sólo 2.995pts.**

De venta en quioscos, grandes superficies y tiendas especializadas

Subiendo al podium

Los mejores videojuegos de la histor

PacMan, Tetris, Street Fighter y Resident Evil sólo son algunos de los juegos que han hecho historia en el mundo de los videojuegos. A pesar de pertenecer a géneros tan distintos, todos ellos tenían algo en común que les ha hecho triunfar en el mundo del ocio.

Pero qué es lo que ha hecho que estos juegos se hayan subido al podium? Quizás nadie tenga la respuesta pero la verdad es que todos nosotros nos hemos desvivido por alguno de ellos en alguna de nuestras horas delante de la consola, el ordenador e incluso de las recreativas.

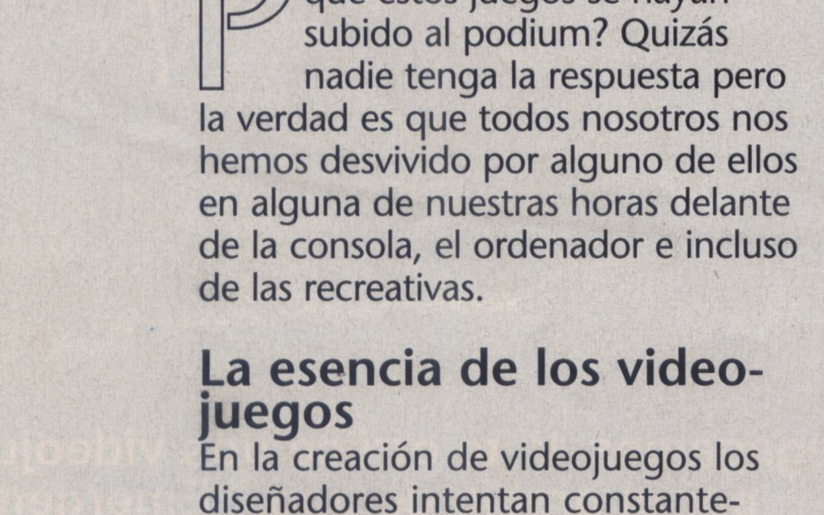
La esencia de los videojuegos

En la creación de videojuegos los diseñadores intentan constantemente potenciar los aspectos de sus productos que provoquen en el jugador la mayor diversión y entretenimiento. La jugabilidad puede ser definida como todos aquellos aspectos del videojuego en los que el jugador interactúa con el programa. Cada uno de esos aspectos hacen al juego único y la innovación en estos campos es materia de profundo estudio por parte de muchas compañías.

¿Podría ser entonces que todo lo que necesitamos para que nuestro juego triunfe se base, en gran parte, en esos pequeños o grandes detalles referentes a la jugabilidad de un producto?

Los juegos son una experiencia global que incluye la jugabilidad, los efectos visuales y sonoros, los personajes, la puesta en escena y el argumento. Mientras que a unos jugadores les puede interesar la puesta en escena y los gráficos, a otros les puede interesar más los personajes y la historia, pero aún así todos los jugadores están de acuerdo en que si un producto no tiene una alta jugabilidad, ese juego no vale la pena.

Para entender qué elementos debemos tener en cuenta para que nuestro juego sea un éxito es necesario extraer la substancia de los videojuegos, quitar todo aquello que sobra y quedarse con la idea principal del videojuego: su esencia.



Starcraft forma parte de un género que mezcla con estilo dos tipos de esencia.

Remontándonos a los viejos tiempos

La manera más fácil de empezar a descubrir qué es lo que necesita nuestro juego para triunfar es rememorar a las viejas glorias que llenaron nuestras horas de ocio en el pasado y que, todavía aún, revivimos de vez en cuando en nuestro recuerdo. Antes no se contaba con la tecnología actual y aún así había juegos que causaban furor en los salones recreativos.

Pongamos por ejemplo el clásico juego *PacMan*. En este título tan entrañable controlábamos a un personaje que simplemente era una gran boca a través de un laberinto tragando grandes cantidades de bolas mientras esquivábamos fantasmas de colores. El juego tenía una gran sencillez, pero el concepto estaba ya en esa pequeña pantalla. Lo que todo el mundo recuerda de ese juego era la adicción que se creaba en esquivar los fantasmas a través de las pantallas, unos fantasmas con un comportamiento ligeramente diferente en cada uno de ellos. Es más, todos recordamos el nudo que se nos hacía en la garganta cuando, apresados por los fantasmas, no podíamos hacer otra cosa que ver como nuestro personaje desaparecía con un sonido simple pero efectivo. La solución a nuestra congoja pasaba por echar otra monedita a la voraz máquina para calmar por un rato nuestra adicción.

Imaginémonos por un momento este juego sin sonido y con una bola blanca en lugar de la animación del comococos y unas bolas rojas en lugar de los fantasmas. El juego sería exactamente lo mismo, pero se habría perdido su esencia. Sin la posibilidad de identificarnos con el protagonista y sin el sonido que se producía al tragar las bolas, nos encontraríamos con un juego sistemático, sin gracia, en el que lo único que veríamos sería movimientos bastante incomprensibles por la pantalla de la máquina recreativa.

Si analizamos detenidamente cada uno de los juegos que han causado furor en el reciente comienzo de la historia de los videojuegos, podremos comprender mejor que es lo que buscan los jugadores en

The Need for Speed, un juego cuya esencia se ajusta muy bien a las reglas.

En los simuladores deportivos la competición es la esencia de su jugabilidad.

En un título de estrategia acrecentaremos nuestro ego construyendo unidades.

cuanto a jugabilidad. Debemos tener en cuenta que esto no es otra manera de describir los videojuegos, sino que en realidad es algo inherente en todos ellos y que tiene una forma mucho menos física que lo que pueden ser los gráficos, los personajes o el guión. Por ello debemos esforzarnos en comprender la idea para conseguir que nuestro juego llegue a triunfar y comparta el ansiado podio de los ganadores.

Buscando un sistema

Mucha gente se introdujo en los videojuegos con títulos como *Asteroids*, *Donkey Kong*, *Space Invaders* o con sagas tan épicas como las de *Tir Na Nong*, *Underwulde* o *Every One's a Wally*. Cada uno de estos juegos ofrecía algo que ansiábamos en nuestra imaginación: pilotar una nave espacial, rescatar una princesa mientras sorteábamos peligros y un largo etcétera que no podríamos acabar nunca. Todos estos juegos nos presentaban diferentes razones para que los probáramos, pero todos ellos se caracterizaban también por su enorme jugabilidad. Cada uno de ellos presentaba unas normas y una forma de controlar el personaje que permitía al jugador triunfar sobre la máquina solamente si eran usadas de la forma adecuada.

Así pues, en estos casos, la jugabilidad se basaba en la dificultad de las reglas y por ello éstas se volvieron progresivamente más y más difíciles, superando a las del juego anterior. Finalmente, llegó el momento en que solamente aquellos jugadores que dominaban las reglas a la perfección podían divertirse y mucha gente se encontró con muchos problemas a la hora de jugar.

Así pues, la primera esencia que nos encontramos en los videojuegos se basa en las reglas, un elemento común a todos los géneros ya sean de lucha, simulación o deportes.

Pongamos un ejemplo en un juego actual como puede ser *Need for Speed* en cualquiera de sus versiones. Al jugador se le da el control y una serie de reglas básicas para simular la conducción de un coche y todas las reglas físicas que la envuelven. Mientras que el objetivo del jugador es derrotar a los otros competidores y conseguir los mejores tiempos en los circuitos, la esencia y jugabilidad se basan en el manejo del coche por los recorridos tan rápido como sea posible, para lo cual el jugador deberá aprender las reglas implementadas para ello. Este aprendizaje es lo que hace divertido los juegos de coches: todos los modelos y circuitos disponibles para el jugador simplemente producen un incentivo extra en el entorno para que el jugador acabe controlando la carrera.

Matando dragones, pero aprendiendo.

Hay muchos juegos que no se centran en las reglas, sino que se centran en el personaje y su condición en el juego. Quizás el ejemplo más claro lo podemos encontrar en los RPGs (Role Playing Games) como pueden ser *Heroes of The Lance*, la serie *Eye of the Beholder* o toda la saga *Ultima*.

Escojamos por ejemplo *Ultima Online*. En este juego podemos realizar una gran diversidad de acciones y hay una gran cantidad de reglas que debemos considerar y aprender antes de poder jugar, pero la esencia es en realidad la posibilidad de que el jugador desarrolle su personaje aumentando su posición y nivel en la sociedad. Comparando ambos aspectos, múltiple cantidad de acciones a realizar y aumento de puntos de personaje, podemos comprobar que difieren totalmente entre sí; pero, igualmente, podemos ver que no son incompatibles, es más, que ambas son necesarias para el perfecto desarrollo de este juego que ha causado furor en la red.

> El aprendizaje de las reglas del juego, la dificultad creciente pero adecuada, entretendrán al usuario

El crecimiento personal en los videojuegos no se basa simplemente en el avance de niveles o la ganancia de experiencia, sino que también podemos incluir otras metas como la de construir alguna

Monkey Island es un verdadero ejemplo de cómo hacer triunfar una aventura gráfica.

cosa, como podemos ver en juegos tan actuales como *Dungeon Keeper* o juegos clásicos como *Sim City*.

Sim City es un juego sin una meta concreta y con un sistema de reglas muy liberal. Para algunos *Sim City* es un juego donde el objetivo es construir la mejor urbe y para otros es construir una ciudad organizada sin crimen ni problemas de tráfico. A pesar de ello todos estos objetivos se basan en el desarrollo de un concepto que tiene el jugador en mente y el sistema presentado es la herramienta de la que dispone el jugador para desarrollarla. Todas y cada una de las decisiones y acciones que tome el jugador repercutirán en el crecimiento de su idea y en este caso de la ciudad.

La competitividad, contra uno mismo o contra la máquina, harán que el jugador tenga un reto contínuo

El crecimiento personal es una de las esencias más importantes para que nuestro juego triunfe, no solamente desde el punto de vista de conseguir más nivel o una mayor puntuación, ya que simplemente el hecho de que el jugador pueda autosuperarse, ya puede influir en que la persona tenga un crecimiento personal. El objetivo final del juego puede ser muy distinto en todos estos casos y en cada videojuego deberemos encontrar la forma en la que el jugador se intente superar a él mismo o al sistema. La competición, contra uno mismo o contra la máquina, siempre es un punto a favor de cualquier programa de software lúdico.

Los juegos de rol juegan con el motivo del crecimiento personal de nuestro personaje.

Las aventuras gráficas nos llevan a un mundo imaginario donde hay que conseguir cumplir los objetivos.

Las ratas y el queso

Tras haber comprendido bien las dos esencias anteriores, facilidad de manejo o aprendizaje y competitividad, nos encontramos con algo mucho más visible que en los dos casos anteriores.

Pensemos por un momento en un juego tan clásico como puede ser *Monkey Island*, una maravilla en lo que se refiere a aventuras gráficas. En este juego no encontramos ningún tipo de crecimiento personal y no tenemos que seguir unas reglas al pie de la letra para poder conseguir la victoria. En este caso la esencia del juego se basa en la situación en la que se encuentra el protagonista y lo que debe hacer para salir de ella: podríamos considerar que el protagonista es la rata que debe escapar del laberinto para conseguir el queso.

Esta tercera esencia la encontramos mayoritariamente en las aventuras gráficas, aunque también la podemos encontrar en otros géneros como puede ser el caso de *Rockford*, un juego tipo puzzle, o *Heavy of The Magic*, un clásico que mezcla el rol con la aventura.

En las aventuras gráficas el jugador es situado en un entorno y se le da una historia y un guión con un objetivo que teóricamente debe conseguir. El jugador se mueve a través de diversos escenarios consiguiendo objetos y realizando acciones para lograr su objetivo final. Los objetos no tienen mayor importancia que permitir realizar acciones, con lo cual ni el personaje ni el jugador tienen un crecimiento personal al conseguirlos, así pues podemos comprobar que realmente se trata de una esencia diferente, ya que no se requiere de ningún tipo de habilidad ni de ninguna regla especial para realizar esas acciones.

El objetivo de las aventuras gráficas es normalmente disfrutar de la historia, de la situación y de los diálogos, ya sea a través de historias cómicas, sistemas inteligentes basados en la resolución de puzzles y un gran etcétera de modelos variados con una esencia común. Si tomamos la idea como la de escapar de un laberinto quizás no parezca demasiado atractiva, pero la esencia es simplemente la manera de que todo lo demás quede unido.

En Faraón debemos construir todo un imperio partiendo de la nada más absoluta.

El Cocktail.

Hemos repasado quizás las principales esencias de los juegos, pero frecuentemente podemos encontrar mezclas de todas ellas para que los jugadores tengan más cosas a las que prestar atención.

Un ejemplo muy claro lo encontramos hoy en día en los videojuegos de estrategia en tiempo real. Este género mezcla las esencias de las reglas y del crecimiento personal. Podríamos poner cientos de ejemplos como el clásico *Dune* o títulos más modernos

**s nos llevan a
o donde hay
r los objetivos.**

ontrar en
puede ser el
juego tipo
ne Magic, un
rol con la

gráficas el
n un entorno
a y un guión
teóricamen-
jugador se
versos esce-
objetos y
ara lograr
objetos no
ancia que
nes, con lo
el jugador
personal al
podemos
ente se
diferente, ya
ningún
e ninguna
lizar esas

venturas
te disfrutar
uación y de
ravés de
mas inteli-
esolución
tcétera de
una esencia
idea como
erinto qui-
do atracti-
implemen-
do lo

las princi-
egos, pero
os encon-
las para
n más
tención.
o lo
en los
en tiem-
cla las
el creci-
nos poner
o el clási-
dernos

como *Starcraft*, juegos donde la esencia principal es la del crecimiento, construyendo sus ejércitos y defensas, pero incluyendo al mismo tiempo una serie de reglas que controlan las unidades: como pueden atacar, cuantos pueden ir juntos, cuanto cuestan, etc... Tanto el crecimiento, de nuestros edificios y unidades, como el aprendizaje y dominio de las reglas de combate, son esenciales para conseguir la victoria de nuestro bando y la victoria sobre el enemigo (competitividad).

Cuando mezclamos estas esencias, si queremos que nuestro juego triunfe, debemos pensar en cómo el jugador va a pasar de una a otra y cómo los consumidores van a aceptar el sistema. Es muy importante tener en cuenta que debemos crear sobretodo un producto jugable, sin resultar extremadamente difícil su uso.

Messiah innova en la capacidad que tiene el angelito protagonista para poseer cuerpos.

El siglo XXI

Habiendo entrado en el nuevo milenio nos podemos preguntar si el desarrollo de los videojuegos se ha estancado definitivamente en estas tres esencias. ¿Realmente la programación no puede dar más de sí? ¿Podemos encasillar algún juego fuera de estas tres esencias?

Quizás la mejor manera de conseguir que nuestro juego triunfe es encontrar la originalidad. *Tetris*, *Resident Evil*, *Commandos*, *Final Fantasy*, etcétera son juegos que triunfaron por una razón muy simple. Innovaron sistemas que hasta ahora habían sido sumamente repetitivos.

Si miramos detenidamente qué es lo que queda por tratar en los juegos, se nos puede ocurrir con cierta facilidad que todos ellos tienen, en cualquiera de sus facetas, dosis de irrealidad: el comportamiento de los personajes es fijo, todo es demasiado estático, etc... Viendo esto, podríamos fijar nuestra vista en una de las técnicas que está siendo estudiada con más detenimiento en la informática hoy en día, no solamente en el campo de los videojuegos, sino también en otros campos como pueden ser la robótica o la aeronáutica.

Estamos hablando, lógicamente, de la inteligencia artificial. Quizás esta sea la cuarta esencia que estamos buscando, un tipo de juegos donde podamos simular completamente otra identidad, la pureza en un juego de Rol, la completa libertad en una carrera de coches y quién sabe cuantas cosas más.

El acercamiento a lo que sería la vida real podría ser pues esa cuarta esencia que revolucionará el mundo de los videojuegos: ¿tener a Bishop en tu casa? , realmente sería algo que vendería mucho.

Los juegos mezclan varias características para enganchar al jugador.

Punto y final

¿Pero, y todo esto de lo que hemos hablado para qué sirve?

La respuesta es sencilla. Si queremos que nuestro juego triunfe debemos comprender en que se basa realmente la jugabilidad, de esta manera será más fácil diseñar aquello que queremos y que los jugadores quieren.

Entender lo que son los juegos desde su base nos ayuda a comprender en su totalidad qué es lo que se espera de los programas y nos ayudará a hacer nuestros juegos mucho más interesantes y divertidos.

> Solucionar situaciones a base de usar el ingenio es otra forma de enganchar a la gente a un juego

¿Pero, tan sólo con eso voy a subir al podium?

La verdad es que no. No nos vamos a engañar, con esto solo hemos descubierto la punta del iceberg que guarda en su interior el secreto del éxito. Solamente un par de consejos finales para que vuestro juego sea mucho mejor: intentad estar un paso por delante de lo que podéis encontrar en cada momento y cread cuantas innovaciones os sean posibles, vuestro juego ganará en calidad y prestigio.

Jordi Martín Caballero

Casos prácticos de DIV2

"Hágalo usted mismo"

Este mes iremos viendo casos prácticos sobre las novedades de DIV 2. Estos casos prácticos se podrían denominar también con el nombre de "trucos". Sea cual sea el nombre (algunos lectores ya conocerán algunos de ellos, otros no) siempre será bueno hacer un repaso, por si acaso.

Bueno, vamos a empezar, pero antes daremos una pequeña explicación de la organización del artículo. Lo dividiremos en varias partes, como viene siendo habitual en esta sección, pasando por cada uno de los aspectos novedosos de esta nueva versión de DIV. Iremos viendo el entorno, pasando por el generador de sprites, el editor, etc. Para luego pasar al lenguaje, haciendo una revisión de todas las funciones, con casos prácticos. Pero, sin mas preámbulos, pasemos a ver los casos prácticos o "trucos" sobre las novedades de DIV2.

El entorno

Empezamos con el entorno, y en concreto con los browser. Cuando se tienen que incluir sonidos en un juego, o aplicación, esta tarea se deja normalmente para el final. Normalmente se tienen que buscar sonidos para los distintos eventos, o por parte del programador, recibirlos del encargado de sonido. Tanto para buscar, como para planificar todos los sonidos, algo muy útil son los browser de sonido, disponibles en el nuevo DIV2.

Anteriormente con DIV1, debías coger algún programa que te permitiera oír sonidos y luego seleccionarlos, copiarlos y convertirlos si era necesario. Ahora, si pulsamos en el botón de prueba cuando se quiera cargar un sonido, lo escucharemos antes de cargarlo en memoria. Con esta opción, podremos ir oyendo todos los sonidos, ya que el programa esperará a que se pulse el botón "aceptar". Otro browser interesante en este mismo aspecto es el de pcx, map. En este caso, se debe pulsar sobre el botón "imagen". Esto también se puede hacer en los FPG, Aunque, este ultimo aspecto, es el de más frecuente uso entre los usuarios. Esta opción, dentro del browser de mapas gráficos, es muy útil a la hora de buscar una imagen dentro del árbol de directorios.

Tanto en el caso de los mapas, como en el del sonido, se puede hacer lo que comúnmente se llama *tagear*, que consiste en poder cargar varios archivos a la vez. Con la tecla "control", y pulsando encima de varios archivos, sin dejar de pulsarla, se consigue seleccionar varios de ellos. Si pulsamos encima de un archivo, y luego con la tecla "mayúsculas" pulsada, se pulsa sobre otro, se seleccionarán todos los archivos que estén entre estos. Estas pulsaciones las conocerán muchos usuarios, pues son las mismas que se usan en Windows.

Otra novedad, que puede permanecer oculta para muchos usuarios, es la de imprimir listados de programa y listas de los gráficos que hay dentro de un FPG. Dentro de los menús "programa" y "ficheros", se encuentran las opciones para poder hacerlo. En el caso de los programas, también se puede imprimir sólo una parte que hayamos seleccionado.

Generador de sprites

En esta sección sólo daremos un truco que consiste en usar una textura de un solo color. Es decir, a la hora de hacer la textura haremos una que sea un punto de un color determinado. Por ejemplo, usando un amarillo, o un gris, se consigue una serie de "Terminators" muy interesantes. Este truco sirve de poco, pero es muy curioso, además de entretenido, ver al policía de la película en su aspecto original.

Editor de mapas gráficos

Otra sección "uni-truco", y se trata también de texturas. Pulsando sobre el botón marcado con la letra "u", se podrá seleccionar un mapa que tengamos dentro del escritorio para usarlo como textura, para poder pintar con las herramientas dentro del editor. El truco consiste en dibujar una textura de pequeño tamaño, por ejemplo con el dibujo de unos ladrillos. Luego en otro mapa más grande, podremos dibujar paredes enteras. Por ejemplo, seleccionando la herramienta "cuadrado" y luego el botón "textura", se elige el ladrillo, y se pinta la pared.

Esto es muy útil también para introducir mapas gráficos dentro de un FPG. Se dibuja un cuadrado sin rellenar, dejando un pixel vacío en el lado derecho, y abajo. Luego se selecciona el botón de texturas y se pinta un gran mapa, donde vamos a trasladar todos los gráficos. El cuadrado debe ser lo suficientemente grande como para que entren todos los gráficos.

Editor de mapas 3D

Y seguimos hablando de texturas, en este caso, de las que se colocan en el suelo. Se puede conseguir meter un mapa más grande dentro del suelo de un nivel 3D. Es decir, coger un mapa de 1024x1000 y usarlo como suelo dentro de un mapa 3D, de modo 8. Para conseguirlo, lo primero que hay que hacer es cuadrar las coordenadas con un múltiplo de dos, como 64,128 ó 256. Luego se divide el mapa original, en distintos sub-mapas, del tamaño que hayamos elegido.

En el editor, se hacen habitaciones, del tamaño del mapa elegido. Lo mejor es empezar en las coordenadas 0,0 de suelo, pero si esto no es posible, se hará en un múltiplo, del tamaño del submapa que hayamos elegido. Se deben hacer tantas habitaciones, como trozos de mapa. Luego se designará a cada habitación la textura que le corresponda. Habrá que girarla antes, haciendo varias pruebas se conseguirá cuadrarla, tanto en coordenadas como en el ángulo necesario de giro. Este truco puede ser muy útil para juegos de coches o para campos de futbol.

El lenguaje

Ahora pasaremos a ver trucos dentro del lenguaje. En muchas ocasiones, no será más que un repaso a las funciones disponibles, en otras serán trucos más prácticos.

Opciones de compilación

Un truco que puede ser bastante provechoso, pero que a la vez es peligroso usar cuando hagamos la compilación, es la opción "Ignore errors". Para activarla, al inicio del programa, se debe indicar la siguiente línea:

```
COMPILER_OPTIONS
_Ignore_errors;
```

Con esto se conseguirá que, si DIV encuentra algún error durante la ejecución del programa, lo ignore. Esto es un arma de doble filo, ya que los efectos que puede producir son imprevisibles, incluso que DIV se quede colgado y se deba reiniciar de nuevo.

Las rutinas 3D

Bueno, en este caso tenemos varios trucos, y revisión de todas las funciones. El primer truco está basado en las banderas. Y consiste, a la hora de hacer un mapa, en usar banderas para posicionar al jugador y a otros elementos. Con esto se evitarán muchos problemas a la hora de programar, y el diseñador de niveles tendrá más libertad a la hora de dejar suelta su creatividad.

El siguiente ejemplo, más que un truco consiste en un aviso. Algo que hay que tener muy en cuenta a la hora de programar un modo 8 es el peligro de salirse del mapa. Si esto ocurre se produce un error.

Y seguimos hablando de texturas. Esta vez de animarlas dentro del mundo, en un modo 8, con programación. Estas texturas pueden ser las del suelo, con lo que se pueden conseguir lagos, como en el ejemplo de ski que viene con DIV2. También se puede usar este truco para poner televisiones o relojes dentro de las paredes del mapa y después animarlas. Las funciones necesarias se verán más adelante, aunque lo que hay que tener claro es el número de sector y, en su caso, de pared si fuera necesario cambiar algo.

Seguimos con otro consejo de precaución. En este caso es el de unir dos o más vértices juntos, de varias habitaciones distintas. En este caso, es fácil, que se produzcan errores a la hora de visualizar el mapa, sobre todo cuando se ve una habitación desde otra. El mejor consejo para evitar problemas es el de utilizar el método de prueba-error. Mover un vértice y a veces eliminar, si fuese necesario.

Bueno ahora pasemos a ver todas las funciones.

```
load_wld("nombre_fichero",numero_fpg)
start_mode8(ide_camera,num_mode8,num_region)
stop_mode8();
```

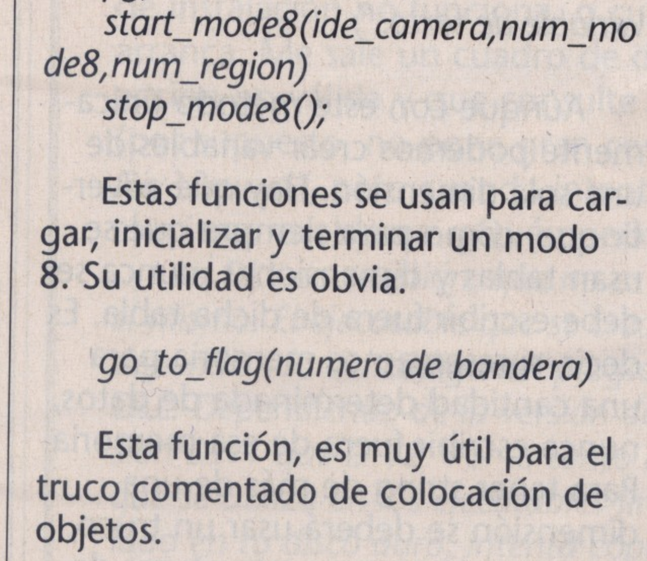
Estas funciones se usan para cargar, inicializar y terminar un modo 8. Su utilidad es obvia.

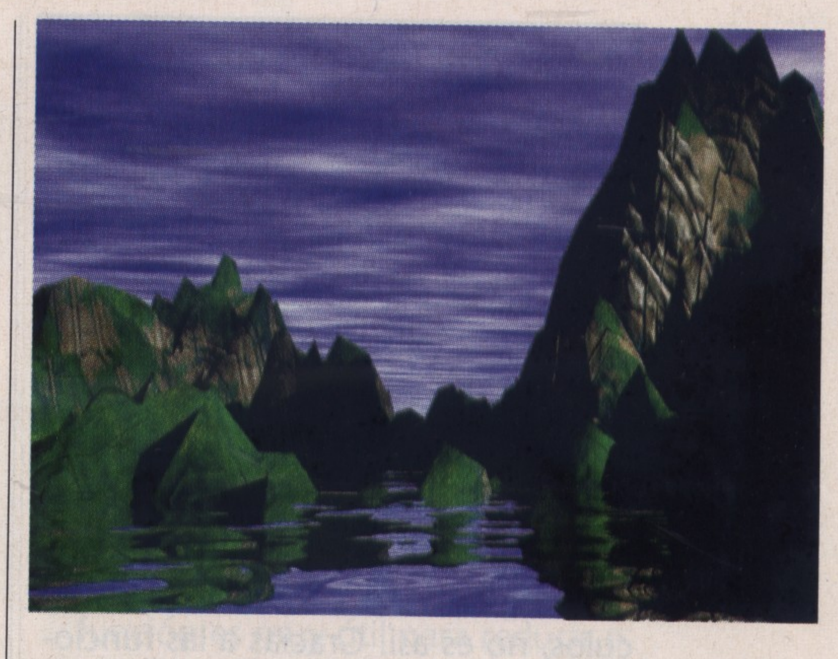
```
go_to_flag(numero de bandera)
```

Esta función, es muy útil para el truco comentado de colocación de objetos.

```
set_sector_height(sector,suelo (-1 sin tocar),techo(-1 sin tocar))
get_sector_height(sector,&suelo,&techo)
```

Estas dos funciones devuelven y modifican las alturas de una zona de





modo 8, muy útiles para abrir y cerrar puertas de hoja, o para los ascensores.

```
int set_point_m8 (punto,x,y)
int get_point_m8 (punto,&x,&y)
```

Si en vez de hojas que se abran subiendo o bajando, queremos que se muevan , o para mover sectores, se puede usar estados funciones que modifican y devuelven las coordenadas de un vértice de modo 8.

```
set_fog (%inicio(-1 sin tocar),
%fin(-1 sin tocar))
set_env_color (%r,%g,%b)
```

En un modo 8, se puede usar una especie de niebla, con estas dos funciones se podrá elegir la distancia de visión, y el color de la niebla.

```
set_sector_texture (sector,suelo
(-1 sin tocar),techo(-1 sin tocar),fade(-
1 sin tocar)(0-15))
get_sector_texture (sector,&suelo,
&techo,&fade)
```

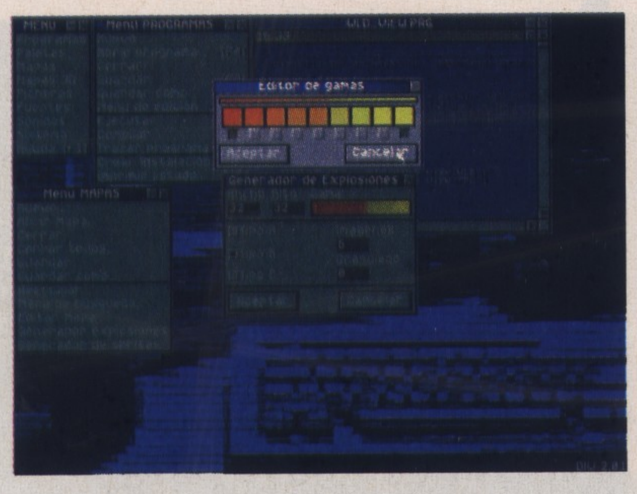
Gracias a estas dos funciones, podremos usar el truco que antes comentábamos, de animar una textura de suelo o techo, dentro de un modo 8.

```
set_wall_texture (pared,textura(-1
sin tocar),fade(-1 sin tocar)(0-15))
get_wall_texture (pared,
&textura,&fade)
```

Por ultimo, las correspondientes al cambio de texturas de las paredes. En todas las funciones anteriores, cuando la función empiece por *get*, será para recoger los valores que estén en ese momento en uso. Cuando empiece por *set*, se utilizará para designar dichos valores.

Buscando caminos

Las rutinas que se llaman de "Inteligencia Artificial", o para abreviar IA, que se han implementado dentro de DIV2, no son otras que las de búsqueda de caminos. Aunque parezca algo simple, el que



un objeto llegue de un punto de inicio hasta otro final rebasando obstáculos, no es así. Gracias a las funciones disponibles podemos conseguir una serie de coordenadas que permitan que el objeto en cuestión llegue a su meta, rebasando todos los obstáculos que se encuentren. En un momento dado, sería posible mediante estas funciones sacar a un objeto de dentro de un laberinto. También existen funciones como las que nos permiten conocer o no si hay un camino libre realmente entre los dos puntos. O la que permite conocer si se puede ir en línea recta. Pero pasemos a ver un caso practico, haciendo un mapa de durezas, usando las funciones, etc.

Lo primero es saber por dónde se va a mover el objeto, es decir el escenario. En dicho escenario, se debe usar un tono especial que haga de color. Para hacer esto de forma cómoda cogeremos el mapa original, y lo pasaremos a mapa de grises. Luego, con un color vivo, como puede ser el azul, el rojo o el amarillo, pintaremos las paredes del mapa. Una vez pintado, eligiremos la opción "mapa de busqueda", dentro del menú "mapas", y aparecerá un cuadro, donde se introducirán el tamaño de tile, o tilesize, y el color con el que hemos pintado las paredes dentro del mapa.

Ya tenemos el mapa de búsqueda, que podremos guardar en el disco duro como si de un mapa normal se tratara. Ahora llega la hora de programar, y usar las funciones de búsqueda de caminos. Todas las funciones trabajan desde procesos, ya que las coordenadas *x* e *y* iniciales, se cogerán del proceso que la llame. Además, siempre es bueno comprobar si existe realmente un camino libre entre los dos puntos antes de llamar a la función de búsqueda de caminos en sí. Esta rutina, como parámetros necesita el offset de una estructura y el tamaño de la misma. Lo mejor, para evitar complicaciones, es crear una estructura del tipo:

```
STRUCT puntos[100];
        x,y;
end
```

Otro aspecto importante, de usar este método, es que, como se basa en un mapa gráfico, se pueden cambiar paredes gracias a las funciones como *map_put_pixel()*. También podremos cambiar el modo de búsqueda, entre rápido o lento. Bueno, ahora veremos las funciones una a una:

- *path_find(modo,fichero,codigo,tilesize,x,y,offset,sizeof):* esta función se encarga de buscar el camino en sí.
- *path_line(file,code,tilesize,x,y):* determina si puede ir en línea recta.
- *path_free(file,code,tilesize,x,y):* determina si un punto está libre.

La función más importante es *path_find()*, que es la que se encarga de buscar el camino, y guardar los puntos en la estructura que hayamos creado. Aunque siempre es bueno comprobar antes con *path_free()*, para evitar cálculos innecesarios.

Las rutinas sobre texto

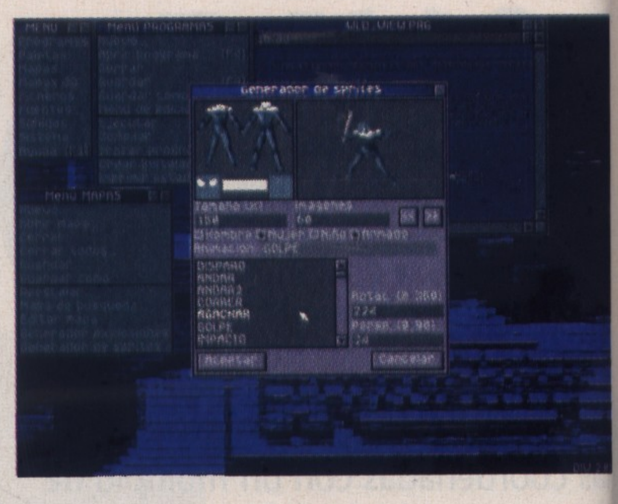
Esta nueva versión de DIV, incorpora muchas rutinas de textos. Además existe un nuevo tipo de datos, llamado "string". Con él podemos guardar cadenas de texto, y manipular su contenido. Un ejemplo de declaración de string, sería el siguiente:

```
STRING texto[100]="Esta es una variable de texto.";
```

Aunque con este método únicamente podemos crear variables de una sola dimensión. Hay que advertir que, como pasa siempre que se usan tablas y dimensiones, nunca se debe escribir fuera de dicha tabla. Es decir, si reservamos memoria para una cantidad determinada de datos, nunca escribir fuera de esa memoria. Para tener string de más de una dimensión se deberá usar un truco, que consiste en usar estructuras de string. Por ejemplo, para tener una tabla con 10 nombres, de un máximo de 20 caracteres, usaríamos la siguiente estructura:

```
STRUCT tabla[9];
        STRING nombre[19];
END
```

Esto se podría usar para tener tres o más dimensiones uniendo más estructuras. Luego, aparte del nuevo dato, tenemos una serie de funciones, muchas de ellas con el mismo nombre que las funciones de texto del lenguaje C. Con dichas funciones podemos copiar, concatenar, medir, comparar, rellenar, pasar a mayúsculas o minúsculas y borrar



caracteres. Al final, veremos todas esas funciones. Aunque una recomendación del que suscribe es que, si una función no nos da el resultado esperado, siempre se puede copiar directamente, usando código ASCII. Por ejemplo, si se quiere introducir una letra A, dentro del primer nombre de la tabla en la primera posición, se podría hacer de la siguiente manera:

```
tabla[0].nombre[0]=65;
```

Aunque siempre se tiene que tener cuidado con el final del string si este cambia con mucha facilidad. En el caso que debamos cortar un string dándole un carácter final, se deberá usar el valor 0. Por ejemplo:

```
tabla[0].nombre[1]=0;
```

De esta forma, se pueden crear string con saltos de líneas y símbolos y códigos que son difíciles de introducir de otra manera.

Este método se puede usar para otros cometidos, como puede ser el de introducir textos desde teclado. Es decir, se recibe una pulsación de una tecla y se guarda en un string con el método antes descrito. A esto se le llama, para reducir términos, un input de teclado. Con lo anterior se ha resuelto el problema de introducir los caracteres recibidos dentro de un string, que luego se puede usar en lo que se quiera. Para recibir datos del teclado, existe la función *key()* y la variable *ascii*. La función es muy óptima en la toma de datos, pero estos no están en formato ASCII, que es el que conviene para nuestro cometido. La variable *ascii* sí almacena los datos en este formato, pero su calidad de respuesta no es muy buena. Por eso se ha creado una función que una a estos dos elementos en uno, teniendo una función muy parecida a *key*, pero que devuelve los valores en ASCII. El proceso que viene a continuación, únicamente funciona con los números del teclado, y las teclas, creando los string con mayúsculas. Se podría mejorar, leyendo también los códigos de mayúsculas, ALT gr, etc. El proceso es el siguiente:


```
process S_ASCII()
private
  CC1;

  TABLA_CONV_ASCII[127]=
    0, 0,49,50,51,   //0..4
   52,53,54,55,56,   //5..9

   57,48, 0, 0, 0,   //10..14
    0,81,87,69,82,   //15..19

   84,89,85,73,79,   //20..24
   80, 0, 0, 0, 0,   //25..29

   65,83,68,70, 71,   //30..34
   72,74,75,76,165,   //35..39

    0, 0, 0, 0,90,   //40..44
   88,67,86,66,78,   //45..49

   77,44,46,45, 0,   //50..54
    0, 0, 0, 0, 0,   //55..59

    0, 0, 0, 0, 0,   //60..64
    0, 0, 0, 0, 0,   //65..69

    0, 0, 0, 0, 0,   //70..74
    0, 0, 0, 0, 0,   //75..79

    0, 0, 0, 0, 0,   //80..84
    0, 0, 0, 0, 0,   //85..89

    0, 0, 0, 0, 0,   //90..94
    0, 0, 0, 0, 0,   //95..99

    0, 0, 0, 0, 0, //100..104
    0, 0, 0, 0, 0, //105..109

    0, 0, 0, 0, 0, //110..114
    0, 0, 0, 0, 0, //115..119

    0, 0, 0, 0, 0, //120..124
   0, 0, 0;        //125..127

 begin
  from cc1=1 TO 127;
   IF (key(cc1))
    return(TABLA_CONV_ASCII
[CC1]);
   END
  END
  return(0);
 end
```

Este proceso se debe usar siempre en un bucle, de la manera siguiente o parecida, incluyendo los demás elementos necesarios para cada aplicación en particular:

```
WHILE (s_ascii()==0)
        FRAME;
END
```

Una vez visto el problema del input, pasaremos a ver las funciones para el manejo de cadenas, esta es la lista completa:

- *string strcpy(cadena1,cadena2):* copia la cadena2 dentro de la cadena1, desde el inicio.
- *string strcat(cadena1,cadena2):* une las dos cadenas, una detrás de otra.
- *int strlen(cadena):* devuelve el número de caracteres de la cadena pasada como parámetros:
- *int strcmp(cadena1,cadena2):* compara las dos cadenas, devuelve un valor de verdadero o falso.
- *int strchr(cadena1,caracteres):* busca los caracteres dentro de la cadena1, devuelve –1, si no encuentra nada, o el índice donde comienza.
- *int strstr(cadena1,cadena2):* esta función hace lo mismo que la anterior, pero se llama de otra manera, es decir busca una cadena dentro de otra.
- *string strset(cadena1,cadena2):* rellena la cadena1, con el carácter dado en la cadena2.
- *string upper(cadena):* convierte a mayúsculas una cadena.
- *string lower(cadena):* convierte a minúsculas una cadena.
- *string strdel(cadena,n_inicio, m_final):* borra n caracteres del inicio y m del final, también admite negativos (añade espacios), esta función es muy útil para reducir o ampliar string.

Resumiendo

Bueno, acabamos por hoy por falta de espacio y por no comenzar otro tema. Dejamos para otro día las rutinas de ficheros, del sonido y de la música, las de las primitivas gráficas y las funciones de memoria. Espero que les saquéis provecho a estos consejos y os espero en el próximo artículo. Si tenéis alguna duda hacérmela saber en tizo@geocities.com. por vía email o a la dirección de la revista.

Antonio Marchal (Tizo)

Preguntas

Tengo un problema: cuando creo una instalación en DIV2, o bien el fichero de instalación no funciona, o cuando funciona, el archivo que instala tampoco arranca. Me sale un cuadro de diálogo que me dice que se ha ejecutado una acción no válida y que consulte a la ayuda para ejecutar archivos en MS-DOS (por supuesto, no pone gran cosa). He pensado que tendría que ver con la *Div32run.dll*, pero cuando intento abrir la *dll* con el juego en cuestión, me sale el mismo problema. ¿Qué puedo hacer? No sé muy bien cómo van las *DLLs*

La DLL en cuestión, que no es otra que DIV32RUN.DLL, es el corazón de DIV en sí mismo. El ejecutable que se crea se encarga de llamar a las distintas funciones dependiendo del código del programa. Todas estas funciones están dentro de esta DLL. Dependiendo de la versión de DIV, la DLL varía, ya que no usan la misma versión DIV1 que DIV2. Por lo tanto esto no tiene nada que ver con tu problema ya que se utiliza en los ejecutables finales. Lo mas seguro es que DIV2 esté mal instalado en tu disco duro, intenta copiar todo otra vez, y sobre todo el directorio INSTALL dentro de donde tengas DIV2.

DIV2 en Internet

Al igual que ocurrió con DIV1, su segunda versión no ha pasado inadvertida en Internet. Ya hay algún juego que saca toda la chicha a la nueva herramienta. También se han visto sus puntos flacos, y ha habido alguna que otra desilusion. En el canal de IRC, siempre suele haber gente. Podéis conectaros al mismo, mediante algún IRC Hispano, como: irc.irc-hispano.org

Luego conectaros al canal #DIV, y ya estáis en contacto con otros usuarios. Aunque para permanecer en contacto también existen listas dedicadas a DIV. La mas importante esta en: www.onelist.com

Su nombre es canaldiv, y hay mas de trescientas personas apuntadas. Un consejo, no te apuntes si no quieres recibir 20, 30 o más mensajes diarios. Esta lista, con tantas personas, tiene mucho movimiento de correo.

Introducción al diseño gráfico

Practicando lo ya aprendido

En este artículo se va a continuar modelando el personaje que, si se recuerda bien, no es orgánico sino al contrario, es poligonal, como los de juegos tan famosos como Quake, Unreal o similares. En estos casos las texturas pueden hacer maravillas ya que incluso con un objeto plano se podría simular relieve o profundidad.

Como ya se comentó en el artículo anterior, la técnica utilizada para modelar cualquier personaje poligonal es empezar modificando vértice a vértice utilizando el modificador "*Edit Mesh*". Es importante que haya quedado claro lo que se ha explicado en artículos anteriores. Por ejemplo, será de gran ayuda saber "*extruir*" las caras (como cuando se modeló la nave espacial). Para que no haya confusiones se va a dar un repaso rápido a cómo "*extruir*" las caras de un objeto, de manera que los lectores que no lo hayan entendido puedan ponerse a la altura de los que lo entendieron a la primera. No obstante, los que ya lo entendieron podrán dar un repaso a sus conocimientos de *3ds MAX*. Pasemos a la practica 1.

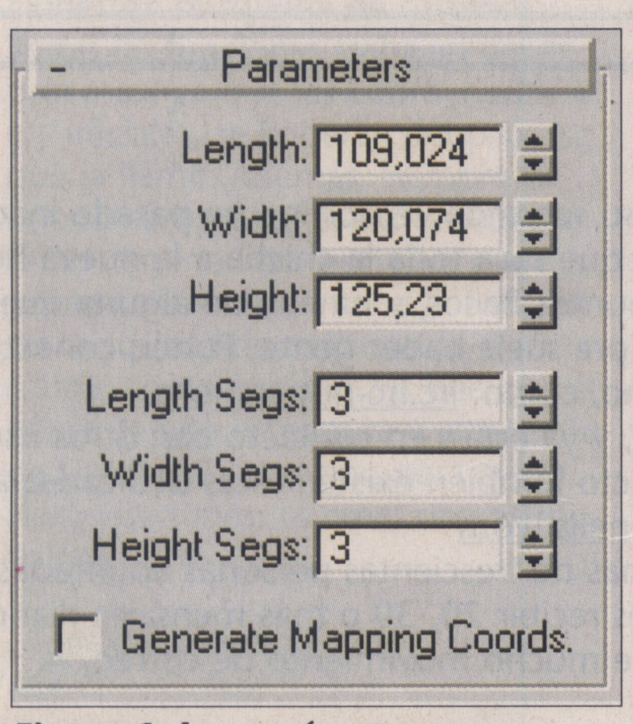


Figura 1: los parámetros.

Práctica 1: extruir caras

En esta práctica no se va a modelar ningún objeto conocido, simplemente vamos a practicar la *extrusion* de caras. Para empezar, crearemos un cubo normal y corriente de la siguiente forma.

En el apartado *create* y en los sub-apartados *Geometry/Standard Primitives* se debe pulsar sobre el botón *BOX*. Lo que conseguiremos con esto es crear un cubo normal y corriente. A continuación se van a aumentar el número de segmentos del cubo, de manera que nos vaya mejor para modificar. En nuestro caso, lo que se hará es dar los siguientes valores:

Width Segs: 3
Length Segs: 3
Height Segs: 3

El resultado de lo que se acaba de hacer debería ser como el de la figura 1.

Una vez tenemos este cubo con tantas caras para modificar es hora de que nos pongamos a trabajar. Lo primero que haremos es extruir una cara central. Se debe escoger uno de los lados del cubo y *extruir* la cara central.

Para ello, lo que se va a hacer es activar el modificador *EditMesh* que está en el panel de modificadores. Si no está entre los botones estándar lo que se deberá hacer es buscarlo entre todos ellos usando el botón que pone *more*. Una vez activado el modificador lo que se deberá hacer es configurar *selection level* como *faces*. De esta manera, lo que se va a hacer es que, cada vez que modifiquemos el objeto, se van a modificar las caras, es decir, que si seleccionamos algo, lo que estaremos haciendo es seleccionar caras. Lo mismo pasa cuando se escale algún objeto. El objeto no será escalado sino sólo las caras seleccionadas. Esta es una forma muy cómoda de modelar.

Una vez que estamos en el apartado "caras", lo que se debe hacer es seleccionar la cara central del lado del cubo que deseemos *extruir*. Al hacer esto, lo que conseguiremos es algo parecido a la figura 2.

Ahora ya se tiene la cara seleccionada por lo que ya se puede *extruir*. Para llevarlo a cabo, lo que se debe hacer es modificar el valor *amount*.

Con ello, lo que se conseguirá será coger esa cara y tirarla hacia dentro o hacia fuera del objeto, de manera que podemos ir modelando una forma. Una vez modificado, el resultado debería ser como el de la figura 3, dependiendo del número hasta el que se ha *extruido* la cara.

Siguiendo este mismo paso que se acaba de explicar, se pueden hacer ya bastantes figuras complejas que nos llevarían más tiempo si se tuvieran que modelar cubo a cubo. Seguidamente se va a explicar como *rotar* las caras para darle a la *extrusión* una inclinación que tienen algunos objetos de la realidad, ya que no todo es ángulos rectos.

Inclinación de las caras

Para inclinar una cara lo que se debe hacer es *rotarla* antes de *extruirla*, de manera que, una vez inclinada, cuando se hace el *extrude*, consigamos el efecto que estamos buscando. Continuando con el objeto que estabamos modelando, volvemos a seleccionar la cara que ya habíamos seleccionado y ahora la *rotamos* con el botón correspondiente de la barra de herramientas.

Una vez inclinada la cara debería quedar como en la figura 4 que muestra la cara que antes se había *extruido*, un poco inclinada antes de hacer una nueva *extrusión*.

Una vez conseguido este efecto lo que se va a hacer es volver a extruir la misma cara y el resultado (muy curioso) es el de la figura 5.

Como es de suponer, las caras también se pueden extruir con un método muy sencillo que es con la cara seleccionada, pulsar el botón de escalar de la barra de herramientas y escalar la supuesta cara.

Un consejo: si se quiere *rotar* o *escalar* una cara de una forma sencilla, lo que se puede hacer es, una vez que tengamos el objeto seleccionado, pulsar *espacio*. Con ello lo que se conseguirá es que, sin tener que pulsar sobre el objeto, podremos afectar a él. Es decir, si queremos escalar la cara (por ejemplo) no la deberemos estar escalando pulsando sobre ella, sino que se podrá estar en una zona vacía de la pantalla, de manera que se podrá observar tranquilamente el efecto que se causa sobre el objeto o cara que se está modificando.

Damos por acabada esta práctica 1, que esperemos haya sido de utilidad para varios de nuestros lectores. Seguidamente vamos a pasar a la práctica 2 que es más sencilla y corta pero también de vital importancia.

Se va a proceder entonces a la práctica 2 que consiste en la edición de vértices mediante el modificador *EditMesh*. Este modificador es de vital importancia para la edición de cualquier tipo de objeto y es uno de los modificadores más extensos y de los que más se utilizan a la hora de modelar formas poligonales.

Práctica 2: edición de vértices

En muchas ocasiones, cuando estamos modelando, no nos basta con la técnica de extruir las caras. Es por ello que también podemos modificar vértice a vértice todo un objeto tridimensional. Esto no se recomienda con los objetos orgánicos, puesto que tienen un sinfín de vértices, pero es muy útil para los personajes poligonales con un número de vértices bastante bajo.

Para poder modificar vértice a vértice se necesita cambiar *selection level* (que antes se ha puesto en *caras*) a *vértices*, tal y como muestra la figura 6.

Con *selection level* de vertex activado, lo que se va a conseguir es lo mismo que si lo ponemos en *faces* (caras), es decir, que todo aquello que seleccionemos quedará afectado si movemos, escalamos o inclinamos.

La herramienta escalar con los vértices es muy útil, ya que va uniendo los vértices seleccionados en un espacio reducido.

Como se puede observar, si tenemos esta opción activada, se pueden ver en los vértices del objeto una especie de estrella.

Se puede hacer que estas estrellas también aparezcan sin usar el modificador *EditMesh*. Esto interesa sólo para saber si la malla tiene muchos vértices o pocos y también es útil a la hora de unir curvas saber dónde están los vértices en los que se unen.

Para ello, lo que se debe hacer, sin estar dentro del modificador, pulsar con el botón derecho sobre el objeto que deseamos y pulsar sobre propiedades. Una vez allí hay un apartado que se llama: "Vertex ticks". Pues bien, basta con activar esta opción para que las estrellas estén en el objeto siempre que lo deseemos. Para desactivar esta opción se debe seguir la misma pauta.

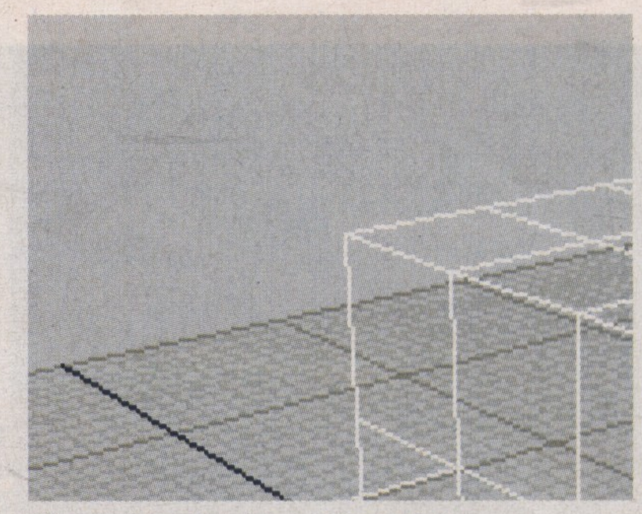


Figura 3: la cara extruida.

Para inclinar una cara lo que se debe hacer es rotarla antes de extruirla

Ahora se va a proceder a la práctica 3. El uso de operaciones *boleanas*.

Práctica 3: operaciones con los objetos

Con la edición de vértices y de caras se pueden hacer objetos

Dudas

Ahora se va a pasar a responder las preguntas de los lectores.

1º Duda:

Estoy empezando con esto del 3D y me gustaría saber más sobre los programas que puedo utilizar. Gracias

Respuesta:

Hay muchos y muy diferentes programas de 3D en el mercado, sin embargo, si quieres seguir este artículo te recomendamos que uses MAX *ya que es bastante fácil de aprender y rápido con las tarjetas convencionales. Pero, ya que lo pides, aquí tienes una lista de los mejores programas de 3D y lo que se ha hecho con ellos.*

En primer lugar podríamos nombrar a Maya, *un programa de Alias Wavefront que se dio a conocer gracias a una animación llamada "Bingo" en la que salía un payaso y... bueno, esta animación se debe ver si se quiere tener una idea del potencial de* Maya *que te aseguro que no es bajo. Puedes entrar a la pagina de Alias Wavefront para más información. A parte de la animación de Bingo se han realizado muchos trabajos (incluso de cine).*

En segundo y último lugar, se debe destacar Softimage, *otro Software que rompe barreras. Este software es muy potente y hace tiempo que está en el mercado. Actualmente existe la versión* 3.8 SP2 *que destaca por encima de otros programas de 3D por su facilidad de uso y su potencia.*

2º Duda:

He oído hablar de una cosa llamada Shaders. Si no me equivoco es lo mismo que plugins, ¿no? Cuál es la diferencia con los Scripts.

Respuesta:

En efecto, Shaders y plugins vienen a ser más o menos lo mismo. En Softimage *se les llama Shaders mientras que en* MAX *se les llama Plugins (cosas de la vida).*

Si se tuvieran que definir se podría decir que son trozos de aplicación implantados en nuestro programa de 3D que lo hace más potente.

La diferencia básica entre los Scripts y los plugins/shaders es básicamente que los plugins/shaders son compilados mientras que los scripts son interpretados.

Si alguien esta interesado en la programación de plugins o scripts para programas de 3D que me escriba un mail a: thorse3d@hotmail.com, ya que existe una lista de correo en español y se está montando una página web sobre ello.

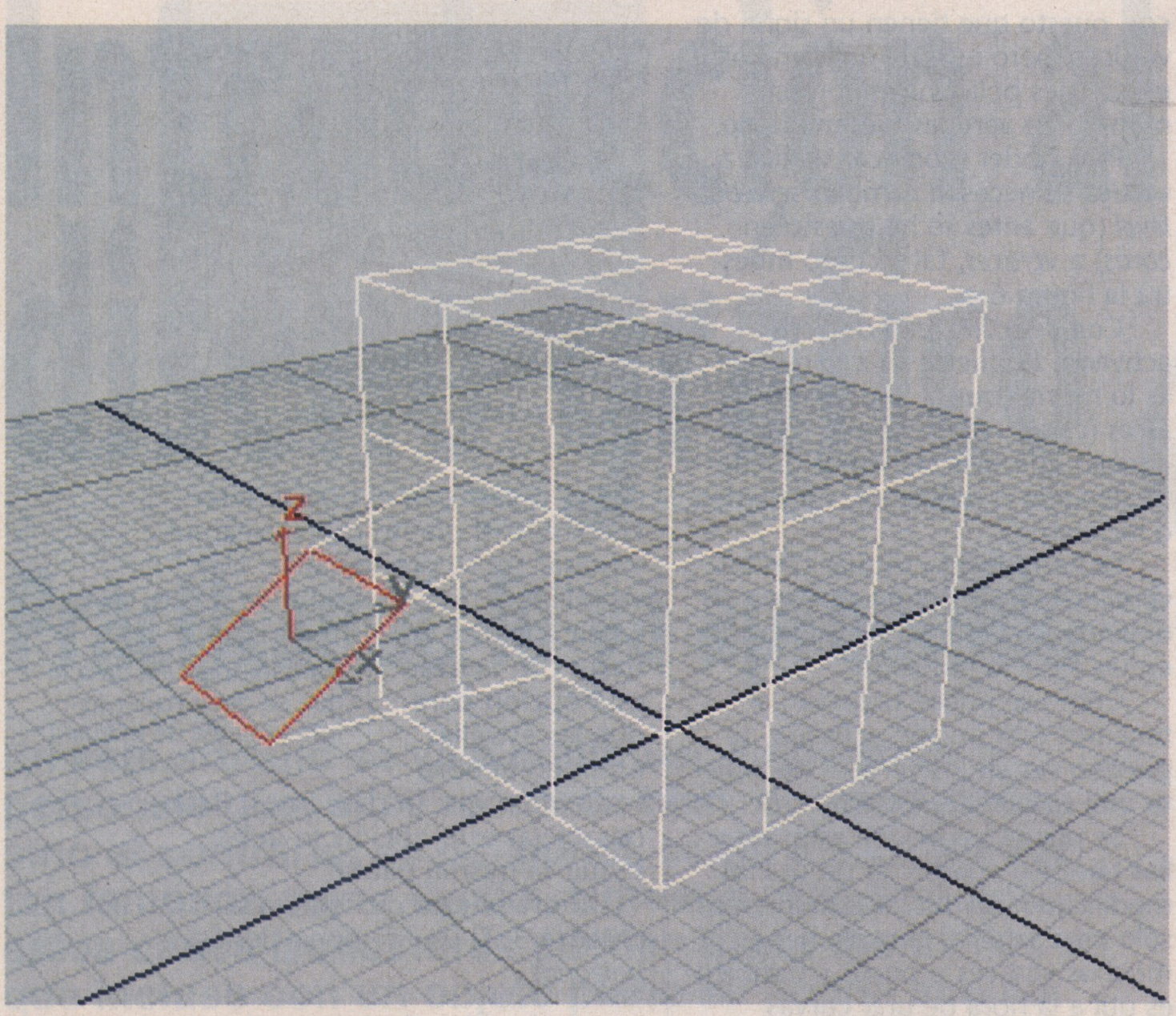


Figura 4: la cara un poco inclinada.

muy conseguidos, pero donde realmente se nota que *3dsMAX* es un buen programa de 3D es en que tiene operaciones booleanas (aunque a veces fallan un poco). Las operaciones booleanas sirven para unir más de un objeto o para perforar un objeto a partir de otro.

Por desgracia, MAX algunas veces falla al hacer operaciones booleanas repetidas

En esta práctica se van a hacer varios objetos. En el primero se podrá observar la potencia de las operaciones de este tipo y la segunda ya será un ejemplo.

Lo primero que se deberá hacer es crear un cubo. Una vez creado creamos un cilindro encima suyo y movemos el cilindro de manera que penetre en el objeto, es decir que lo atraviese.

Nota: es importante que el cilindro atraviese el cubo, de lo contrario, la operación boleana puede salir mal.

Una vez hecho esto se deberá correr por el entorno de *MAX* hasta llegar al siguiente apartado: Create/Geometry/Compound objects.

Ahora se acaba de entrar en un apartado donde hay herramientas que tienen algo en común. Lo que tienen en común estas herramientas es que forman un objeto a partir de otros.

Lo que se debe hacer ahora es seleccionar la herramienta *boolean 2* que nos servirá para hacer las operaciones *booleanas*. Para poder activar esta herramienta es necesario que un objeto este seleccionado, pero no puede ser cualquier objeto sino que debe ser el objeto que va a ser perforado.

Una vez dentro de esta herramienta se deberá seleccionar el botón llamado: "Pick Operand B" y a continuación se debe seleccionar el cilindro. Lo que provocaría esta operación sería un objeto con forma de cubo que tiene una penetración en forma de cilindro.

El autor del artículo recomienda al lector husmear entre los parámetros de esta herramienta, ya sea antes de especificar el segundo operando o después de hacerlo. En el último caso se deberá hacer en el apartado de modificadores, si se hace de esta segunda manera se podrá observar que los objetos de origen no se han perdido y que se pueden mover para corregir el resultado final de la operación que se acaba de realizar.

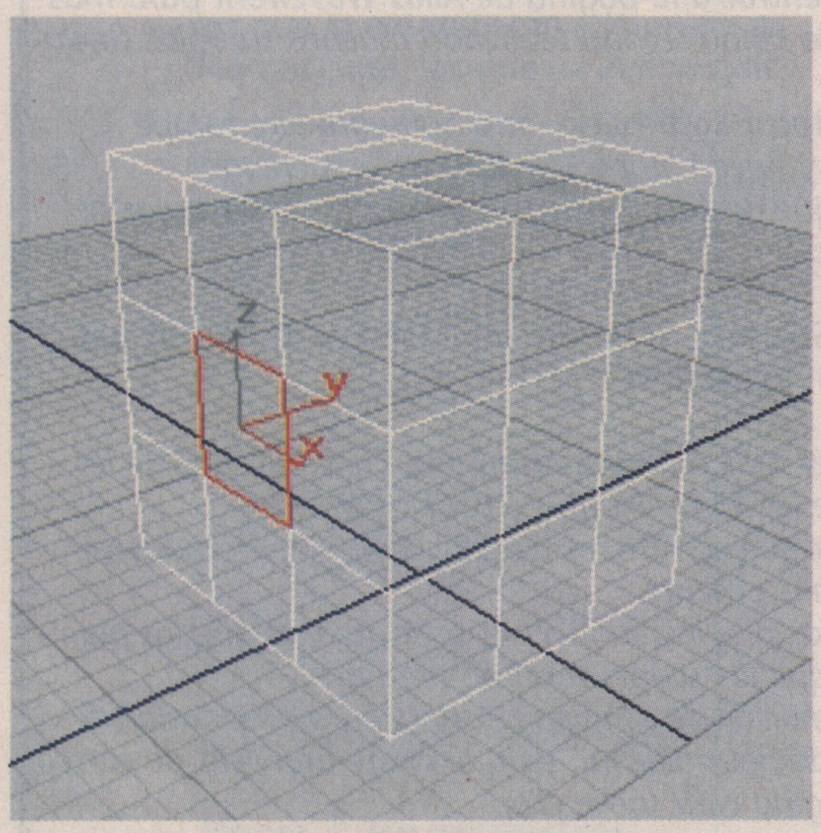


Figura 2: una cara seleccionada.

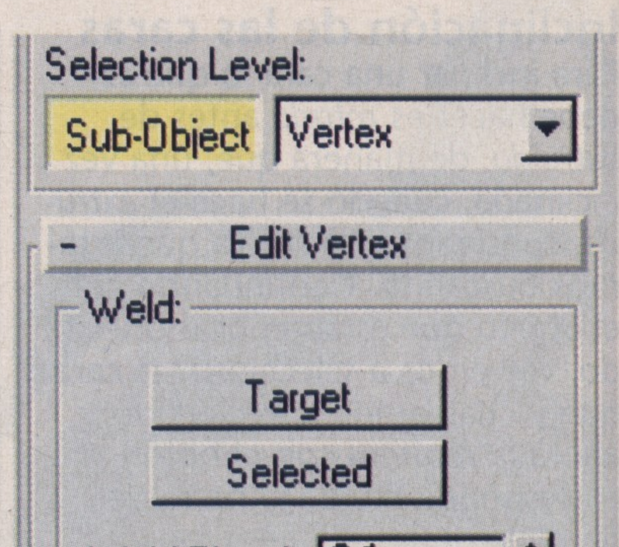


Figura 6: selection level a vertex.

Por desgracia, *MAX* algunas veces falla al hacer operaciones de este tipo si se hacen más de una vez con un mismo objeto. Por defecto, lo que se hace con estas operaciones es perforar, pero también se puede hacer la diferencia de un objeto o hasta unir dos objetos con esta técnica.

Ejercicios

El autor de este artículo recomienda una realización continuada de ejercicios usando estas tres prácticas, ya que es la mejor forma de aprender a modelar en 3D.

En este aspecto el 3D y la programación se parecen, ya que un programador va cogiendo más soltura y se le queda más la teoría cuando, aparte de entenderla tras leerla, intenta hacer cosas.

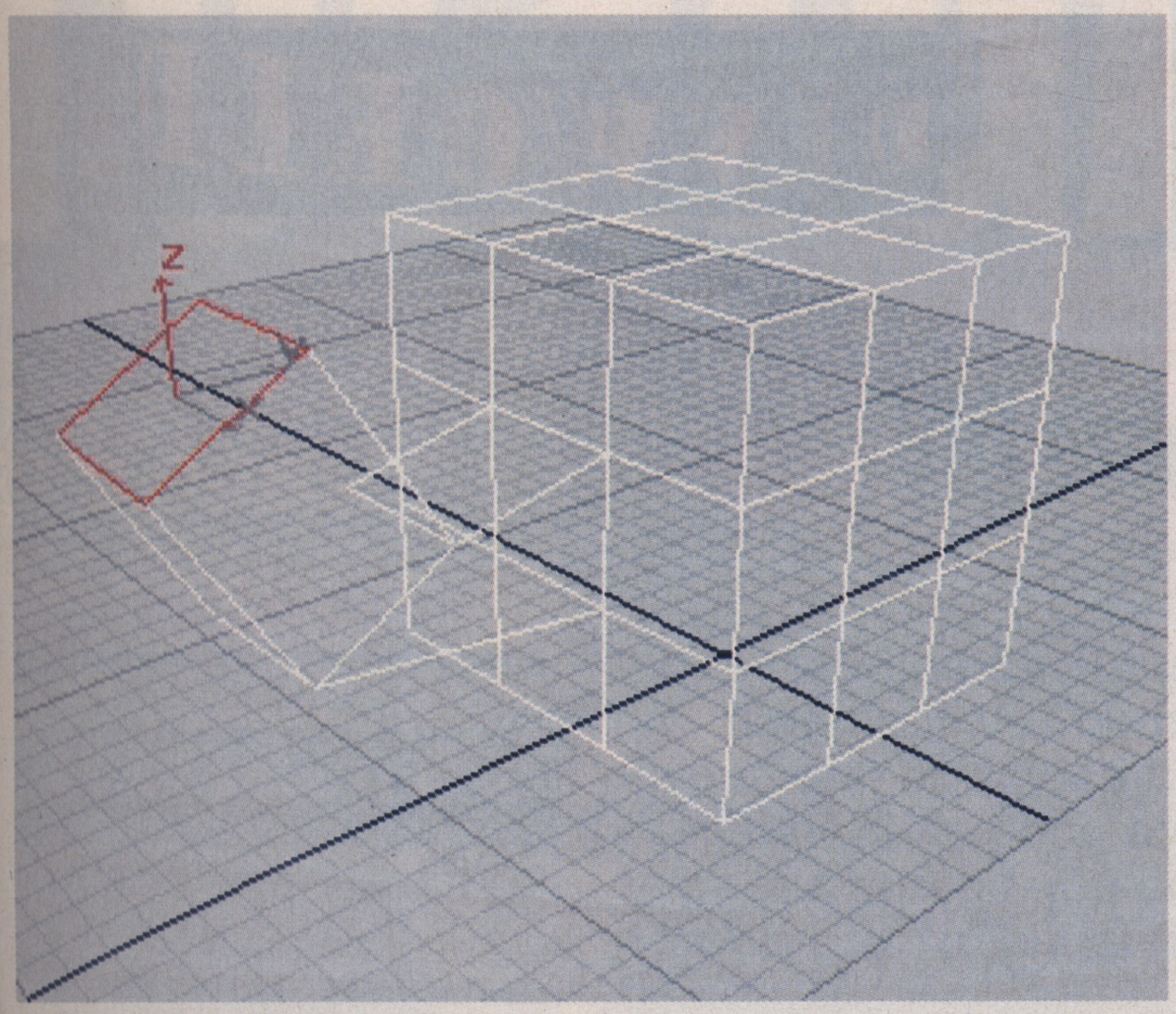


Figura 5: extrusión de la cara inclinada.

Reiteramos que el autor de este artículo está dispuesto a solucionar cualquier problema que surja a la hora de seguir estas líneas o bien de llevar a cabo los ejercicios.

Así pues se van a proponer los siguientes ejercicios para practicar con *3ds MAX*.

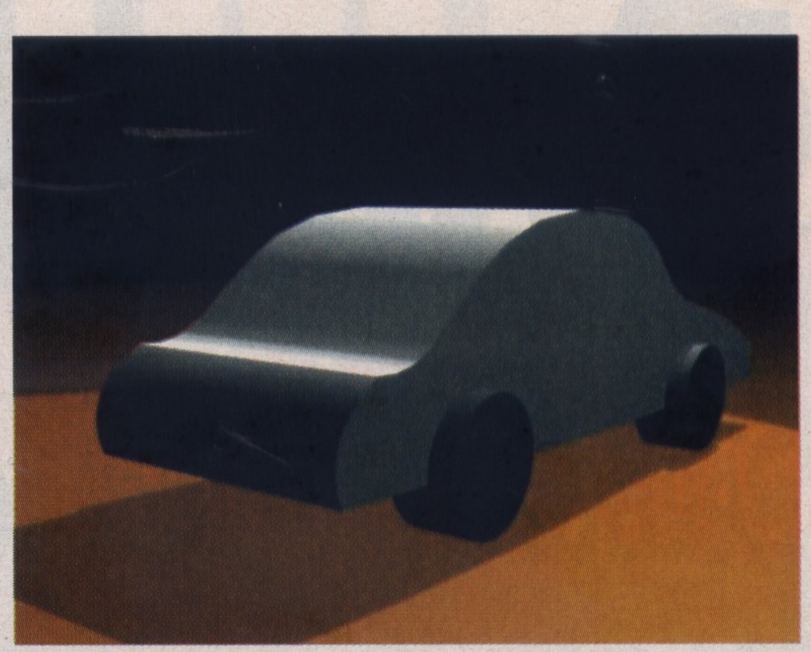
- Crear una nave espacial con la extrusión de caras.
- Creación de un vaso de agua mediante operaciones booleanas.
- Crear la pantalla del ordenador mediante las dos técnicas.
- Creación de un hombre poligonal.

En efecto, la ultima práctica es hacer al hombre poligonal, puesto que ya se ha explicado la teoría. Si se saben hacer los ejercicios anteriores, modelar al personaje orgánico no tiene ningún secreto.

Además los conocimientos de los lectores van mas allá de todo esto, un ejemplo de ello es que se le podría hacer un sombrero al personaje con el modificador *lathle* o *torno* y un poco de edición de vértices.

Un comedor de "diseño".

El sol produce espléndidos reflejos.



Trabajos de nuestros lectores.

Pues bien, una vez más hemos llegado al final del artículo. Como digo al final de todos ellos, esperamos vuestras imágenes en *thorse3d@hotmail*.com y si tenéis algún tipo de duda, sugerencia o consejo, no os cortéis, que el autor del artículo no es ningún monstruo.

Se recomienda encarecidamente que los lectores husmeen por sí mismos en las entrañas de MAX

Es posible que en el próximo artículo se sigan haciendo prácticas, ya que es la forma más fácil de aprender a modelar en 3D. Pero se insiste en que los lectores husmeen por si mismos e intenten hacer cualquier cosa que tengan en mente ya que todo eso es fuente de nuevos conocimientos.
¡Hasta DIVmanía 8!

David Martínez

Envío de imágenes y petición de opinión

Esta vez, no ha escrito mucha gente preguntando cosas, pero si que se han mandado imágenes al buzón del autor del artículo. Gracias a ello se podrá disfrutar de tales imágenes en la revista y el autor va a hacer una valoración personal de ellas, así como algunos consejos para mejorarlas.

Hola. Somos Analizer Studios estamos alojados en www.lanzadera.com/analizerstudios y nos gustaría presentarle nuestras dos últimas producciones en *3D Studio MAX 2.5*. La primera es una representación de la explosión del sol en una ciudad, la segunda es un comedor de una casa simple, la tercera es un diseño que aún estamos desarrollando con NURBS, un coche. Me gustaría que usted me diera su opinión y, si quiere que aparezcan en la revista, adelante.

P.D.1: Los tres diseños han sido creados por Pol Jeremías miembro de Analizer Studios, tengo 16 años y estos son mis primeros trabajos con *3D Studio Max 2.5*.

Felicidades, eres el segundo lector que envía imágenes a mi buzón. Las del primer lector serán publicadas en el siguiente artículo.

Las imágenes están realmente bien, sobre todo me gusta la de la casa pero el coche, ¿cómo lo has hecho?, es decir, ¿lo has extruido? En especial, lo que me gusta del comedor es la iluminación que le has dado a la escena en general y el aspecto de cristal que tienen ciertos objetos.

En fin, sigue así, para tener 16 años no está nada mal. Quién sabe si dentro de un par de años ya eres famoso en el mundo del 3D y eres grafista del juego mas vendido de la historia.

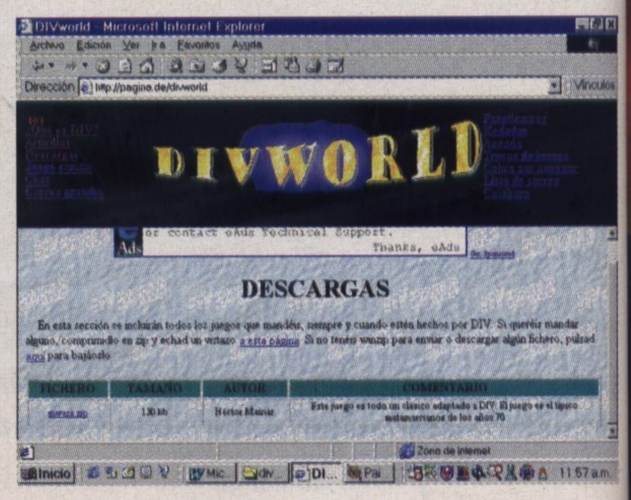
Un pequeño consejo: Si comprimes las imágenes con JPG, intenta que no pierdan tanta calidad, porque imágenes como éstas valen la pena de ser mostradas a los otros lectores.

Sin nada mas que decirte me despido. Espero volver a recibir imágenes de ti o de tu grupo. Esto va también por los otros lectores que a ver si se empiezan a animar a enviar imágenes. Salu2.

Divworld

Revista electrónica sobre DIV

Parece que hay un florecimiento en Internet de páginas dedicadas al mundo DIV, y en esta primavera divera vamos a hablar de una nueva yema que ha brotado hace poco en la Red. Se trata de DIV World, página que se une a la ya esplendorosa floresta que puedes encontrar en la jungla de la Web.

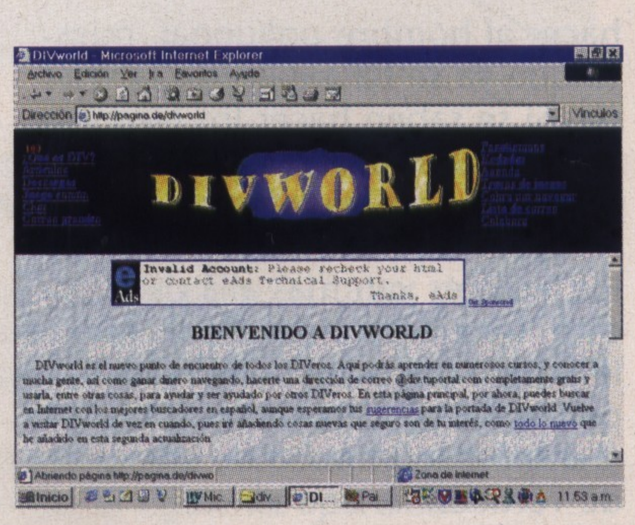


Antes que nada tomad nota de la dirección de esta página, que nos ha llegado vía correo electrónico por mano de su autor, Héctor Mainar: http://pagina.de/divworld. Y una vez terminado este trámite obligado pasamos a ver lo que nos ofrece esta nueva revista electrónica DIV.

Destripando

Después de la presentación de rigor, en la que se nos dice que Divworld pretende ser un nuevo punto de encuentro de todos los DIVeros, se nos hace patente enseguida que esta página, además de informativa pretende ser ante todo útil. La primera prueba es el enlace con cuatro de los buscadores más importantes y usados actualmente: Yahoo, Olé Terra, Ozú y Navegalia Lycos.

En Internet hay cada vez más páginas dedicadas a la programación de videojuegos con el entorno DIV

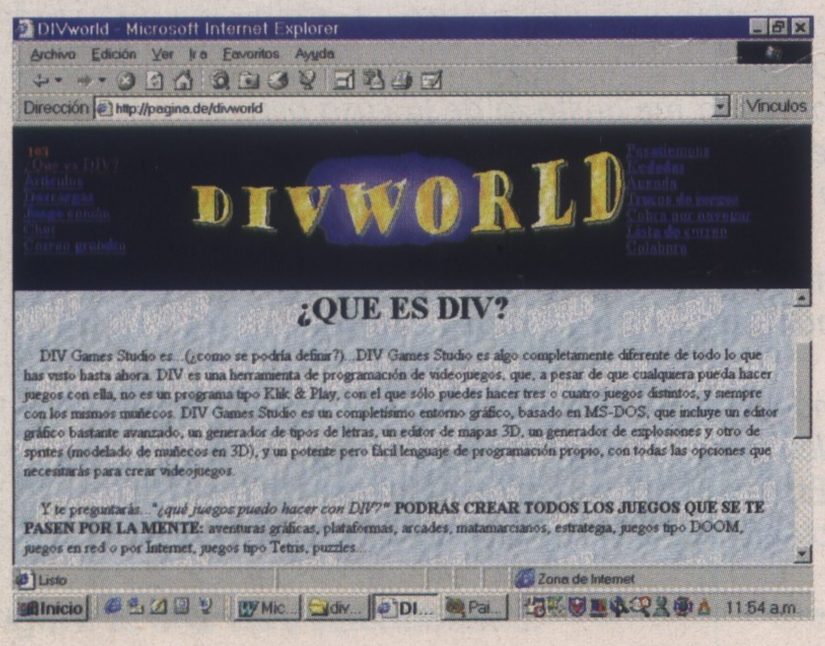


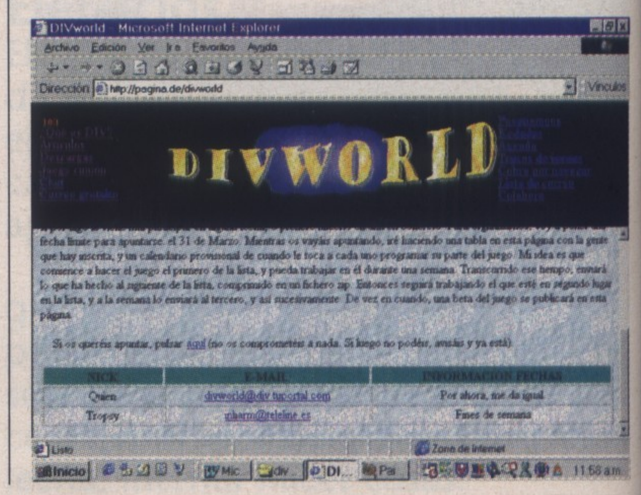
En cuanto al apartado educativo, también son bastante provechosos sus artículos, entre los que podemos encontrar los siguientes:

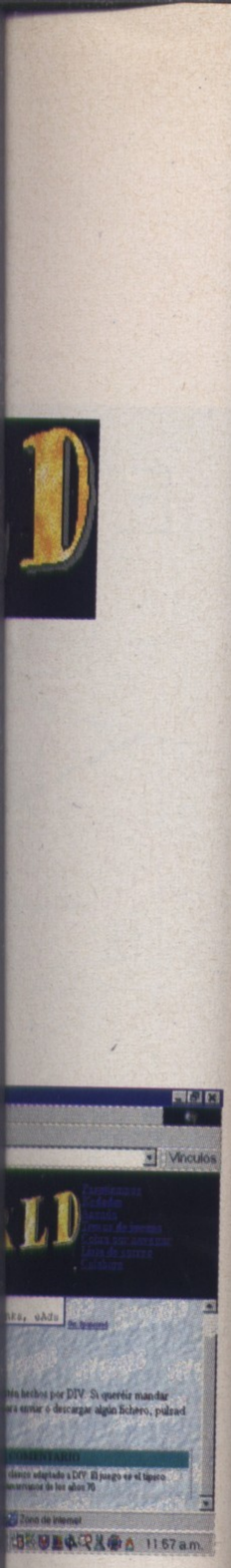
- *El fenómeno DIV*. En el que, después de un breve texto, podremos encontrar un completísimo listado de enlaces, unos 24 sitios para visitar si te quedas con hambre de datos después de visitar esta página.
- *¿Qué es DIV?* En el que se explica qué es esta herramienta de programación de videojuegos, cuáles son sus posibilidades y los requisitos del sistema necesario para usarlo.
- *Iniciación DIV*. La primera parte de este curso para los más novatos consta de dos entregas, la primera dedicada a los procesos, la segunda a los datos. Estas lecciones continuarán en la próxima actualización de la página.
- *Curso de programación de aventuras*. Aquí ya se pasa a algo más práctico, para programar tus propios juegos del género aventurero nada mejor que leer los cuatro temas de los que consta este curso: ¿qué es una aventura?, crear tu propia aventura, definir la aventura y la interfaz gráfica.
- *Curso de IRC*. Para los que se inician en DIV y quieren compartir sus experiencias con otros usuarios aficionados a la programación de videojuegos, nada mejor que utilizar los canales de chat dedicados a DIV. Si somos nuevos usando la Red, en este apartado encontraremos todo lo que hay que saber para chatear con éxito. Lo más interesante, y que seguro interesará a todos, es una completa lista de emoticones, algunos realmente curiosos, como el que expresa que el usuario es un fan de Picaso o un aficionado al submarinismo.
- *Curso sobre HTML*. Para finalizar hay un curso de introducción al lenguaje HTML, en el que podemos enterarnos de multitud de términos usados en él.

Descargas

Como es ya usual en este tipo de páginas, podemos encontrar también algunos juegos realizados con



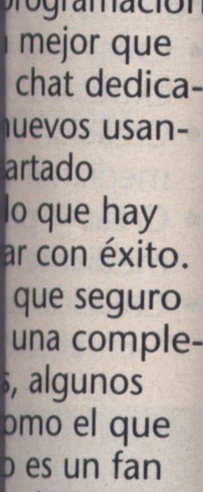




ventura?,
tura, definir la
gráfica.
que se ini-
compartir
otros usua-
programación
mejor que
chat dedica-
nuevos usan-
artado
lo que hay
ar con éxito.
que seguro
una comple-
, algunos
omo el que
es un fan
ado al sub-

ra finalizar
ducción al
que pode-
ultitud de

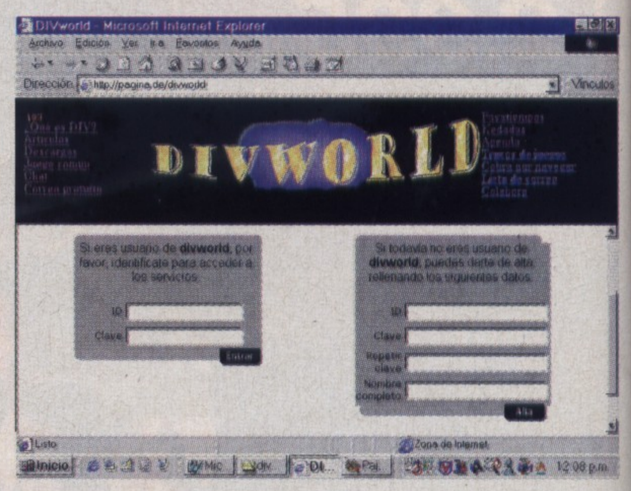
te tipo de
ntrar tam-
lizados con



El test

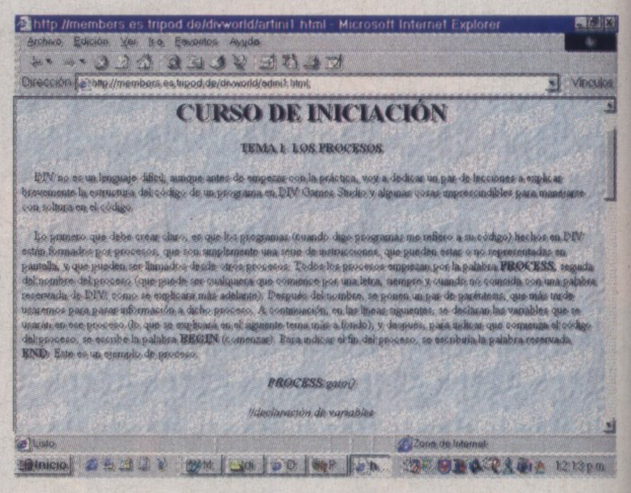
No nos hemos resistido a publicar el simpático test que puedes encontrar en la revista DIV World. Totalmente científico y aprobado por el colegio de psicólogos.
"Este primer test ofrecerá una rigurosa respuesta a la pregunta "¿Es usted adicto a Internet?" Conteste sí o no a las siguientes preguntas. Si todas las respuestas son un rotundo "Sí", probablemente su cerebro acabe convirtiéndose en un módem".

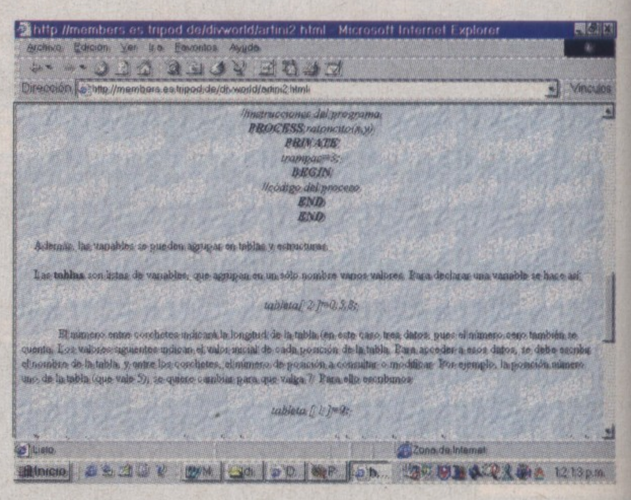
1) ¿Su perro tiene página web propia?
2) ¿Suele besar la home page de su novia?
3) ¿Existe una web marcada en el cristal de sus gafas?
4) ¿Sufre depresiones si no se le ocurren nuevos temas para navegar?
5) ¿Se niega a ir de vacaciones a un lugar sin líneas telefónicas ni electricidad?
6) ¿Va finalmente de vacaciones tras comprar un móvil y un portátil?
7) ¿Sube al avión el portátil y deja a su hijo en el portaequipajes?
8) ¿Escribe "COM" después de cada punto?
9) ¿Se refiere a "ir al baño" como "downloading"?
10) ¿Su corazón se acelera cuando ve una nueva dirección de Internet en la prensa o la TV?
11) ¿Sales de tu habitación y te enteras de que tus padres se han mudado?
12) ¿Ha instalado intercomunicadores por toda la casa para oír cuándo llega un nuevo correo?
13) ¿Ha colgado su esposa una peluca sobre el monitor para que te acuerdes de ella?
14) ¿Todos sus amigos tienen una @ en sus nombres?
15) ¿Todos los links de Internet aparecen morados (ya visitados) en su ordenador?
16) ¿Su bookmark tarda 15 minutos en desfilar?
17) ¿Ha comprado un disco duro de 9 gigas para el historial?
18) ¿Si conservase el historial tendría todo Internet en su ordenador?
19) ¿Ha recorrido todos los links de Altavista y Yahoo?
20) ¿Los ha vuelto a recorrer?
21) ¿Otra vez?
22) ¿No puede llamar a su madre porque ella no tiene módem?
23) ¿Se levanta a mitad de la noche a mirar el correo electrónico?
24) ¿Al ver que no tiene mensajes nuevos pulsa "Actualizar" hasta que aparece alguno?
25) ¿Se refiere a su edad como 3.x ó 4.x?
26) ¿Ha confiscado la línea telefónica de su hija adolescente para la red y ha avisado a sus amigas de que no llamen a ese número nunca más?
27) ¿Sus facturas telefónicas vienen en cajas custodiadas por guardias de seguridad?
28) ¿Cuándo se entera de que va a morir en una semana se las ingenia para mantener en uso su canal de IRC personal?
29) ¿Las tareas del colegio de su hijo las hacen en HTML y las envían por E-MAIL a sus profesores?
30) ¿Ignora el sexo de sus más íntimos amigos porque usan nicks neutros y jamás ha pensado en preguntárselo?
31) ¿Llama a sus hijos "Eudora", "Mirc", "Arroba" y "Puntocom"?
32) ¿Su esposo le cuenta que se dejo barba hace tres meses?
33) ¿Se levanta de madrugada para ir al baño y a la vuelta mira el correo electrónico?
34) ¿Se incendia su edificio y sigue navegando?
35) ¿Dice a los taxistas que vive en http://calle.tal.esquina.tal/51/portal_azul.html?
36) ¿Dice a sus hijos que no pueden jugar al ordenador porque "Papi tiene trabajo que hacer", y usted está en el paro?
37) ¿Se compra un sillón ejecutivo para relax... con teclado y ratón incorporados?
38) ¿Su esposa le ha dicho que "El ordenador no puede venir a la cama"?
39) ¿Cree que los motores de búsqueda son inventos inútiles?
40) ¿Se hace un tatuaje con la frase "Este cuerpo se ve mejor con Netscape 4.0 o superior"?
41) ¿Nunca se desconecta de su servidor?
42) ¿Ha reemplazado la silla de su ordenador por un inodoro?
43) ¿Pregunta a su médico acerca de instalar un giga en su cerebro?
44) ¿Su música preferida es el altavoz del módem?
45) ¿Se maravilla, como ante un milagro, cuando su proveedor le concede "llamadas sin límite" en lugar de las actuales doscientas horas al mes?
46) ¿Su mujer le dijo que la comunicación es fundamental en un matrimonio... y usted le compra un ordenador e instala otra línea telefónica para poder chatear?
47) ¿Cuándo su coche derrapa en una curva y se sale de la calzada en plena montaña, usted comienza a buscar el botón de "Atrás" o "Deshacer"?
48) ¿Es capaz de memorizar cualquier dirección de E-Mail, número IP o dirección de Internet de forma espontánea?
49) ¿Enciende el ordenador y desconecta a su esposa?
50) ¿Comienza a girar la cabeza al sonreír :-)?

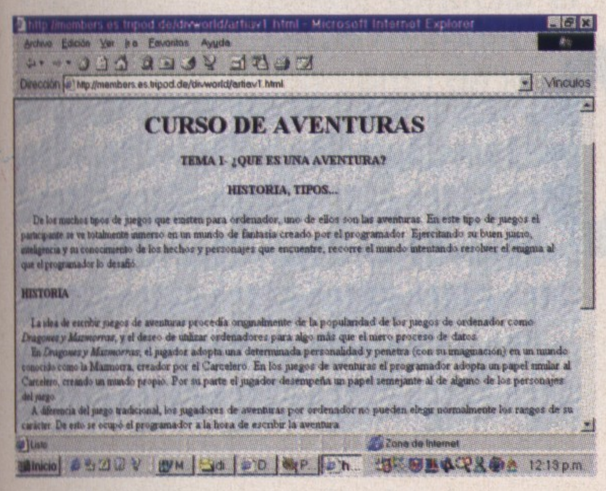


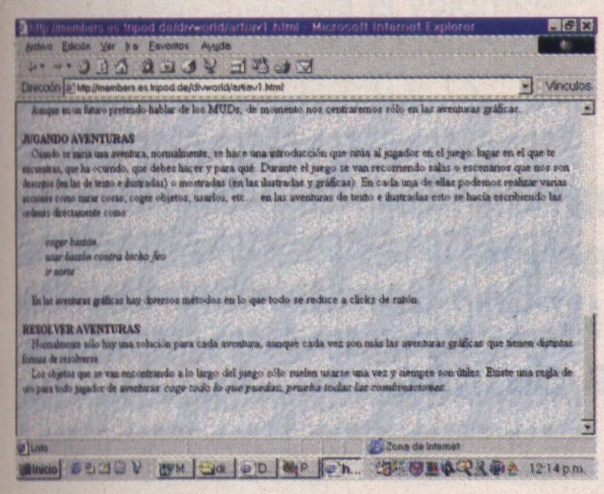
DIV. En este caso se puede bajar un título creado por el autor de la página, se trata de un matamarcianos típico de los años 70 adaptado con DIV.

En cuanto al desarrollo de proyectos futuros, Héctor declara que uno de los objetivos de su página es que todos los aficionados a DIV se conozcan y se ayuden. Para ello propone la creación de un juego entre todos los que se quieran inscribir. Solamente hay que apuntarse poniendo el e-mail y se recibirá la pertinente información para colaborar. La idea inicial es que el primero de la lista trabaje durante una semana en el juego y luego envíe lo ya hecho al siguiente colaborador siguiendo la lista y así sucesivamente. También se tiene pensado poner una beta de lo ya realizado en la página cada cierto tiempo.





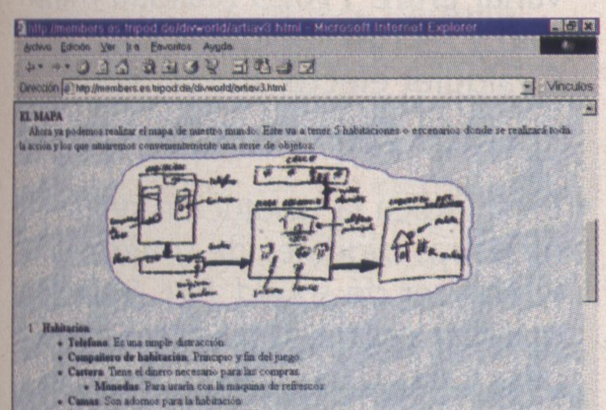


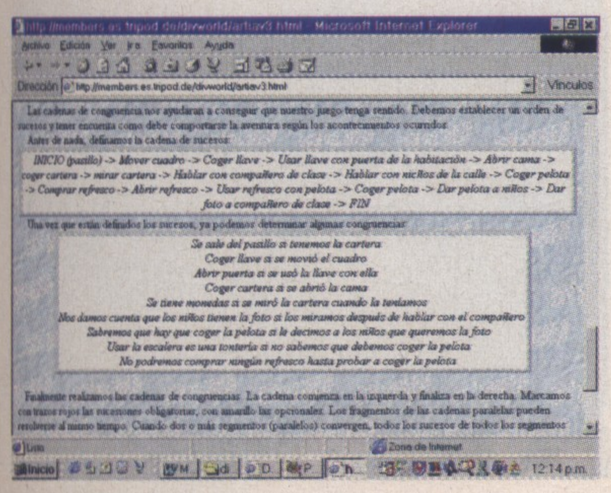


También se intenta realizar una encuesta para ver qué día de la semana es más conveniente para chatear todos juntos. El día más votado se utilizará para establecerlo como cita de todos aquellos que quieran hacer amigos, dar a conocer sus proyectos, colaborar con otros, plantear dudas, etc.

Es posible acceder a una cuenta de correo gratuito y apuntarse a una lista de correo, para estar informado sobre las últimas noticias sobre el apasionante mundo de la programación de videojuegos.

En esta página hay también un rincón para efectuar "kedadas", una agenda, trucos para juegos, y una curiosa sección de pasatiempos francamente entretenida. Para que todos la disfrutéis os ponemos, en los cuadros adjuntos, un curioso test para que midas tu adicción a

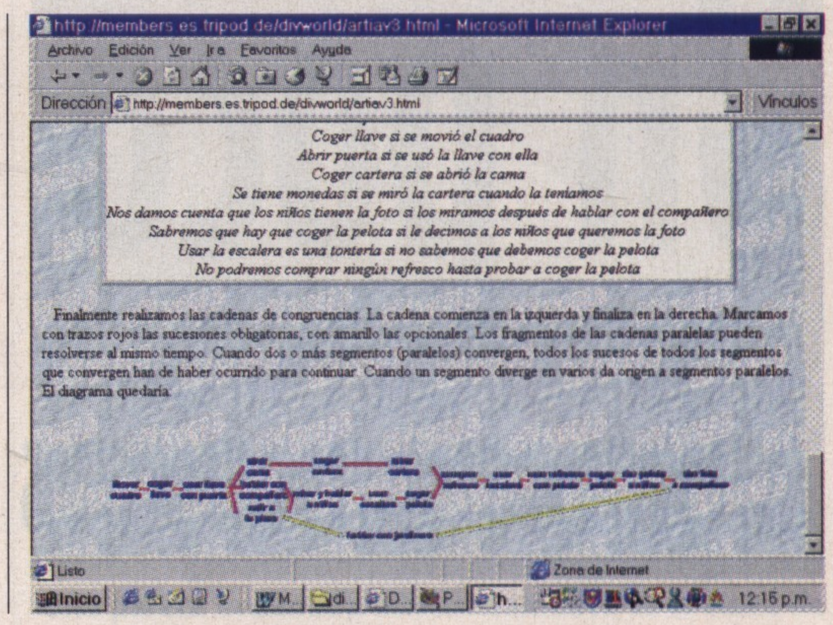


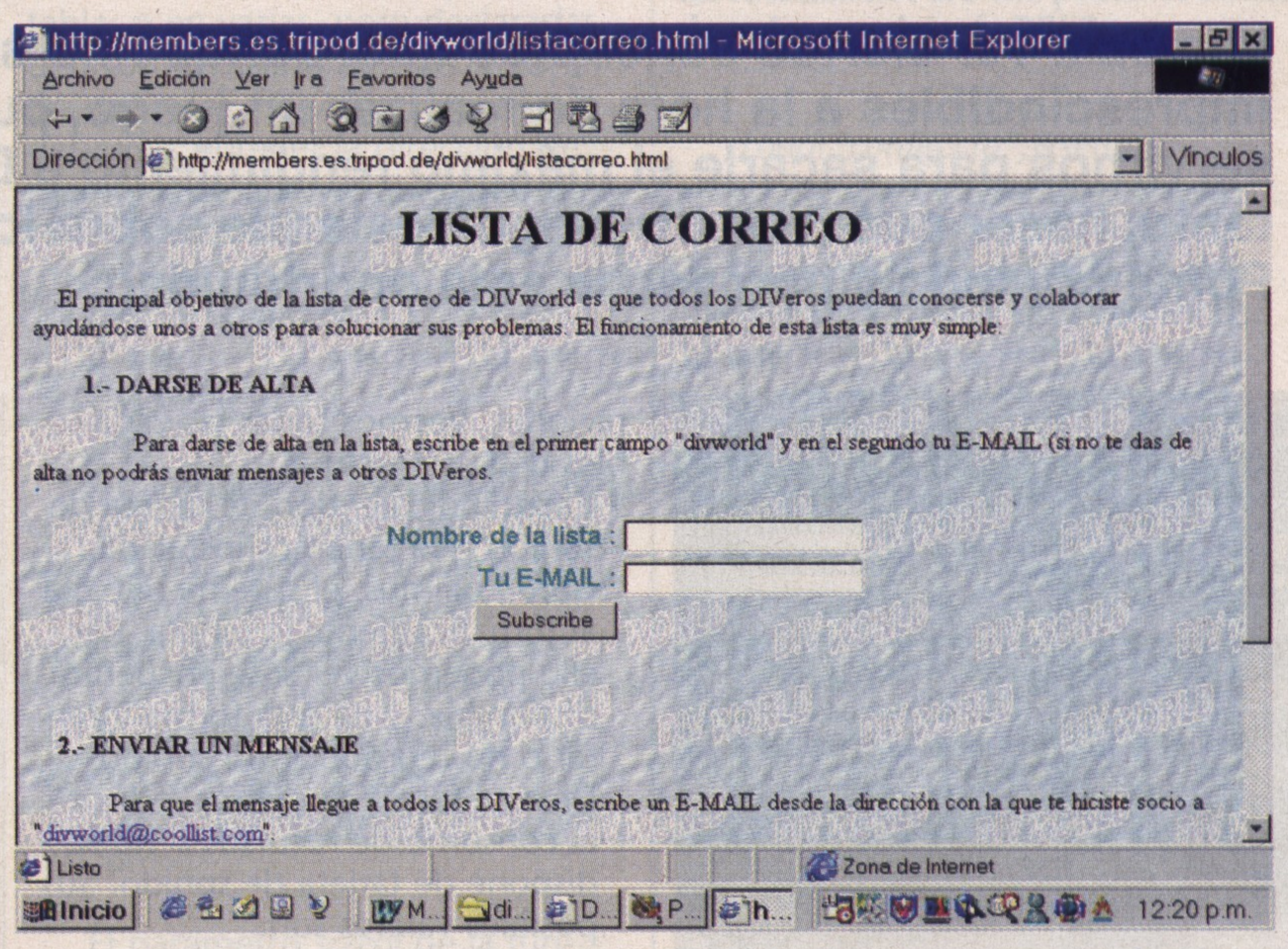


Internet y una sopa de letras acorde con el mundo DIV.

Para terminar este artículo, sólo nos resta decir que todo aquel que quiera ver su página reflejada en esta sección nos puede mandar un correo a la siguiente dirección: gover@prensatecnica.com, ya sea una revista electrónica, un grupo de programación o simplemente un aficionado a DIV. Esperamos recibir noticias vuestras.

Alfredo del Barrio





Sopa de letras

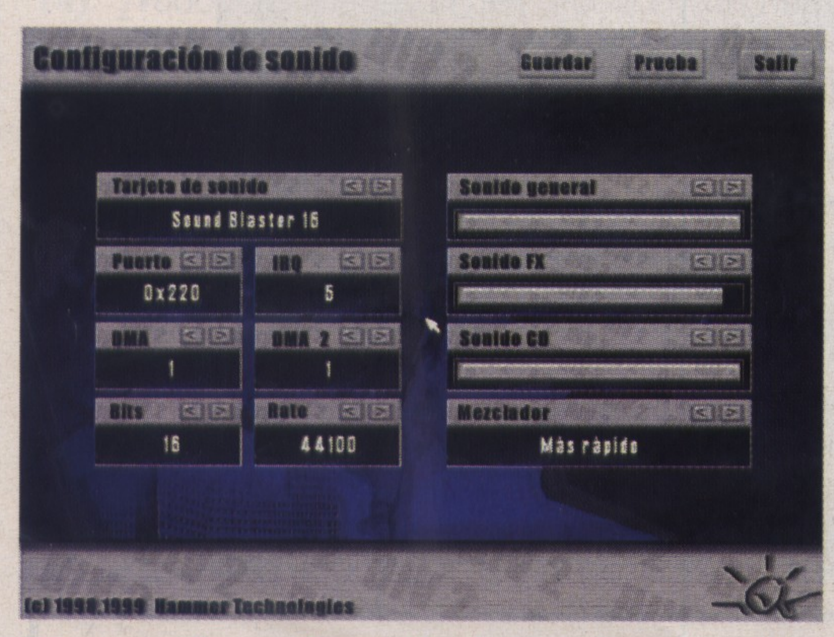
Y de regalo esta sopa de letras que también puedes encontrar en la sección de pasatiempos de esta estupenda página.
Encuentra en esta sopa de letras el nick de once DIVeros.

D	A	E	X	A	E	F	D	M	A	P	M
X	I	M	O	C	E	V	I	C	A	A	F
C	H	Z	F	B	E	A	V	A	Q	T	F
E	Z	E	A	E	A	S	M	N	A	X	G
B	C	Ñ	A	N	A	D	A	E	Ñ	I	Z
I	A	T	D	K	J	Z	S	L	A	A	D
K	T	I	Z	O	I	J	T	F	M	R	C
O	R	O	S	L	D	E	E	M	O	I	R
G	R	E	L	R	E	N	R	T	H	U	T
H	J	E	D	I	V	B	F	I	Z	Y	F
J	R	G	M	A	I	N	A	H	O	S	O

Un poco de todo

Algunos conceptos útiles

Vamos a ver ciertos conceptos que, aunque no sean imprescindibles a la hora de realizar un juego, pueden servirnos para sacarle el máximo rendimiento a DIV y, por tanto, hacer que nuestro juego destaque sobre los demás. Ponte cómodo y aprovecha esta oportunidad.



Programa de configuración de la tarjeta de sonido de DIV. Permite alterar todos los parámetros de la tarjeta.

A lo largo de esta serie de artículos, hemos ido dejando atrás algunos conceptos útiles que ahora vamos a pasar a retomar. Cada uno de ellos tiene toda una serie de utilidades alrededor que sin duda nos servirán. Pasemos sin más dilación a verlos.

Configurando la tarjeta de sonido

Aunque DIV internamente configura el tipo de tarjeta que disponemos a través de la variable de entorno *BLASTER* o *GRAVIS*, según el tipo de tarjeta que tengamos, podemos querer variar estos valores de configuración. Tal vez una utilidad mucho más común y tal vez necesaria es la de variar el volumen de reproducción de efectos de sonido, CD-audio o archivos *MOD* durante la ejecución de un programa. Para todo esto existe una estructura llamada *SETUP*, que contiene todos los parámetros necesarios para realizar estas operaciones. Veamos todas las variables de la estructura:

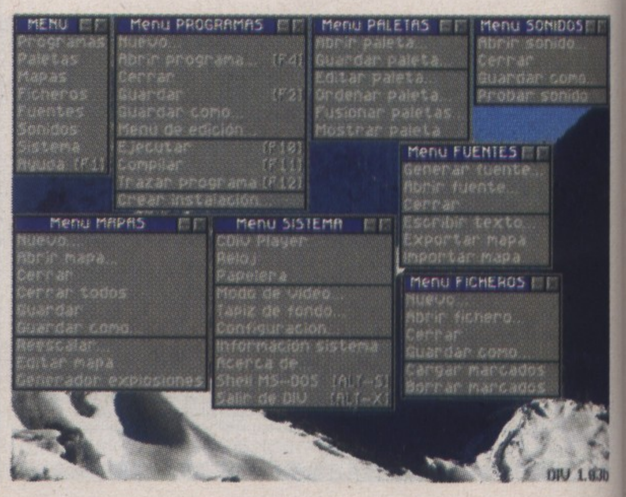
- **Card:** indica el tipo de tarjeta de sonido que tenemos instalada. Como dijimos, el sistema detecta automáticamente la tarjeta. Por ello podemos ver si se encuentra disponible, y de esta forma intentar o no reproducir un sonido. De esta forma podemos reducir el consumo de recursos si no se dispone de tarjeta de sonido. Sus posibles valores son:
  0 = Sin tarjeta
  1 = Sound Blaster 1.5
  2 = Sound Blaster 2.0
  3 = Sound Blaster Pro
  4 = Sound Blaster 16
  5 = Sound Blaster AWE
  6 = Gravis Ultra Sound
  7 = Gravis Ultra Sound MAX

- **Port:** indica el puerto de comunicaciones con la tarjeta de sonido. Su valor más común es 0x220:

  0 = 0x210
  1 = 0x220
  2 = 0x230
  3 = 0x240
  4 = 0x250
  5 = 0x260

- **Irq:** Indica el número de la interrupción o IRQ asociado a la tarjeta de sonido. Sus valores pueden ser (el más frecuente es IRQ 5):
  0 = IRQ 2
  1 = IRQ 3
  2 = IRQ 5
  3 = IRQ 7
  4 = IRQ 10
  5 = IRQ 11
  6 = IRQ 12
  7 = IRQ 13
  8 = IRQ 14
  9 = IRQ 15

- **Dma y Dma2:** ambos corresponden al puerto de acceso directo a memoria o DMA. Sin embargo DMA2 corresponde al puerto de comunicaciones de 16 bits que algunas tarjetas de sonido poseen. Su valor puede variar entre 0 y 10 según el canal utilizado (frecuentemente es el canal 1).
- **Mixer** (sólo DIV2): indica la calidad del mezclador de sonido utilizado por nuestro juego. Su valor puede ser *fast_mixer* para un mezclador rápido, o *quality_mixer* para un mezclador con la máxima calidad de sonido.
- **Rate** (sólo DIV2): indica la máxima velocidad de muestreo de los sonidos a reproducir. Ésta puede variar entre 11025 (calidad mínima) y 44100 (calidad CD). Cualquier efecto que se reproduzca se adaptará a esta frecuencia.
- **Bits:** indica la resolución de las muestras digitales de sonido. Puede ser de *sound_bits_8* bits (calidad pobre) o *sound_bits_16* (alta calidad). Al igual que en el caso anterior, todo sonido se adaptará a esta resolución.



- **Master:** nos permite controlar el volumen general de reproducción tanto del CD como de los efectos sonoros. Su valor puede variar entre 0 y 15, siendo 15 su valor por defecto.
- **Sound_fx:** controla el volumen asignado a los sonidos reproducidos mediante la *función sound()*. Al igual que en el caso anterior, su valor puede oscilar entre 0 y 15.
- **Cd_audio:** controla el volumen de reproducción de CD-audio. También varía entre 0 y 15.

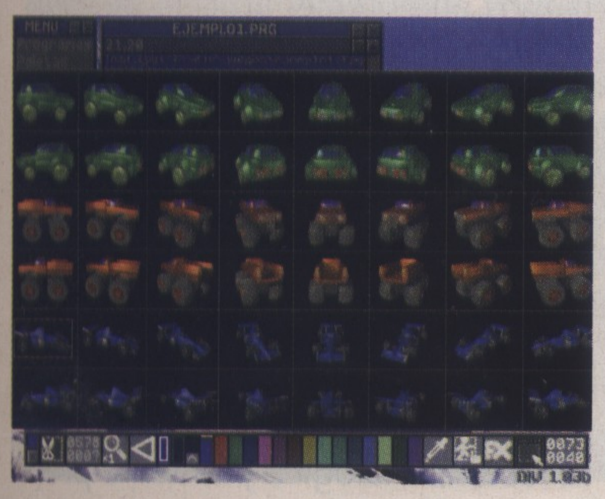
Podemos variar estos valores como nos convenga, si bien, si queremos que tengan efecto debemos ejecutar tras la variación, la función *reset_sound()* si hemos variado algún valor de la configuración de la tarjeta, o a la función *set_volume()* si variamos algún valor de la configuración del volumen.

Además de esta estructura y con relación a la reproducción de sonidos, disponemos de la función *Change_sound()*, que nos permite definir tanto el volumen como la frecuencia de reproducción de un sonido que se encuentra en un determinado canal para su reproducción. Su sintaxis es la que sigue:

Change_sound (<canal>,<volumen>,<frecuencia>);

Donde canal es el número del canal de la tarjeta por el cual se reproduce (DIV permite hasta 16 canales de reproducción), devuelto por la función *sound()* cuando queremos reproducir el sonido; *volumen* es un valor entre 0 y 512 que determina la potencia del sonido; *frecuencia* es un valor también entre 0 y 512 que nos permite cambiar la velocidad de reproducción del sonido, siendo 0 el sonido más grave y 512 el más agudo.

Para poder ver un ejemplo del uso de esta estructura, podemos acudir al archivo SETUP.PRG, del directorio del mismo nombre, y que se utiliza para grabar la configuración de la tarjeta de sonido en el entorno de DIV. Vamos a ver cómo se pueden grabar estos valores y otros cualesquiera en un fichero externo.



Grabando y cargando datos de configuración

DIV dispone de un par de funciones que nos permiten grabar y cargar datos de nuestros juegos. De esta forma podemos grabar nuestras partidas y cargarlas posteriormente, o guardar la configuración entre otras opciones. El único inconveniente es la obligación de almacenar todos estos datos en una sola estructura, variable o tabla; o en varias, pero declaradas dentro del mismo bloque *(GLOBAL,LOCAL o PRIVATE)*. Para ello debemos obtener la dirección de memoria de una variable, que se realiza mediante la sentencia:

OFFSET <variable>
O bien
&variable

No importa en estos momentos el trasfondo de esta sentencia, sino sólo cómo utilizarla en este caso.

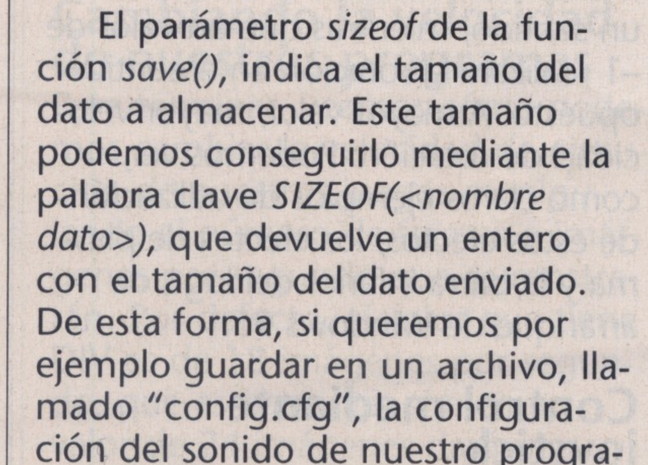
Las funciones de que disponemos son las que siguen:

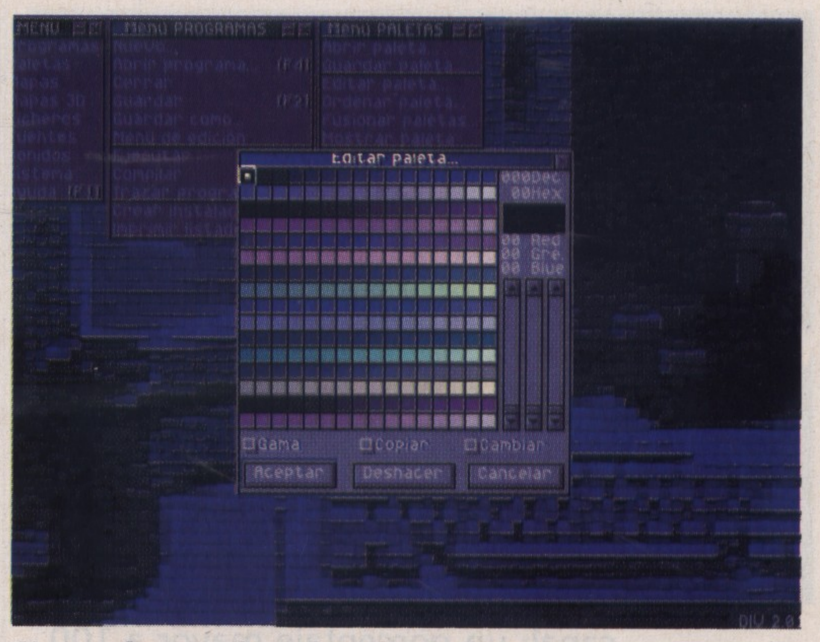
Save(<nombre de archivo>, <offset dato>, <sizeof dato>);
Load(<nombre de archivo>,<offset dato>);

El parámetro *sizeof* de la función *save()*, indica el tamaño del dato a almacenar. Este tamaño podemos conseguirlo mediante la palabra clave *SIZEOF(<nombre dato>)*, que devuelve un entero con el tamaño del dato enviado. De esta forma, si queremos por ejemplo guardar en un archivo, llamado "config.cfg", la configuración del sonido de nuestro programa, sólo debemos escribir la sentencia que sigue:

Save ("config.cfg",&setup,sizeof(setup));

Por analogía con este caso, podemos cargar los datos grabados con anterioridad mediante la función *load*:





Una paleta ordenada adecuadamente es la clave de las rotaciones. Para ello podemos utilizar la opción de ordenar paleta de DIV 2.

Load("config.cfg",&setup);

Si deseamos guardar datos almacenados en más de una variable, debemos tener en cuenta dos cosas:

– Los datos deben encontrarse declarados de forma consecutiva dentro del mismo bloque *GLOBAL, LOCAL* o *PRIVATE*, enviándose como dirección la del primer elemento que en el orden declarado en dicho bloque deseemos grabar.
– Debemos indicar el tamaño de todas las variables que queremos grabar realizando la suma de las sentencias *SIZEOF* de cada una de las variables.

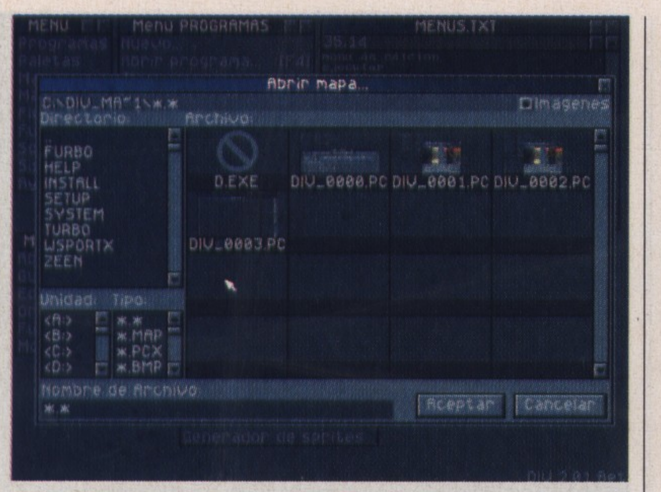
Muchas variables de la tarjeta de sonido pueden ser cambiadas si no se adecuan a lo que queremos

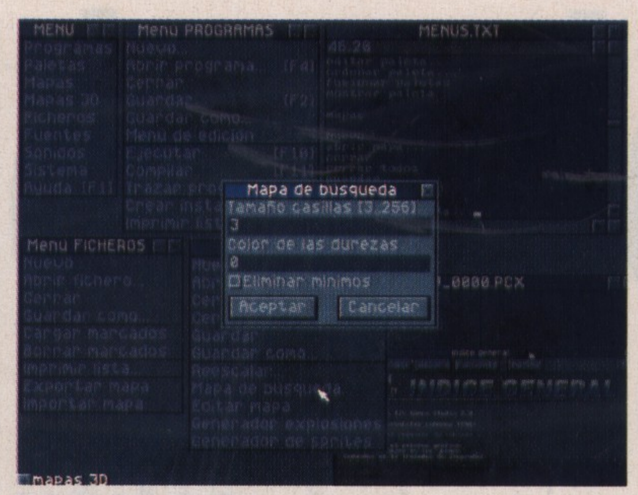
Fundidos de pantalla y otras manipulaciones de la paleta

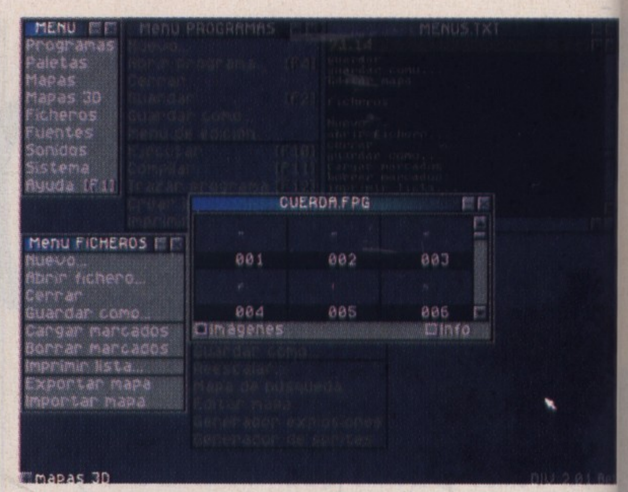
Ya hemos visto cómo realizar fundidos de pantalla simples con las instrucciones *fade_on()* y *fade_off()*. Estas dos instrucciones encendían o apagaban la pantalla, de forma que nos permitiera enlazar dos pantallas distintas de forma suave y dinámica. Pero los fundidos de pantalla en DIV no se limitan a esto, sino que van más allá. Existe una función llamada *fade* que nos permite realizar cualquier tipo de fundido de color y controlar todos los parámetros de éste. Esto además nos da la oportunidad de crear nuevos efectos, como filtros de colores o aumento de la saturación. Veamos en primer lugar la definición de esta función:

Fade (<% rojo>,<% verde>,<% azul>,<velocidad>);

Como podemos ver, cada uno de los canales de color de la paleta puede ser manejado con toda tran-







quilidad. Un porcentaje de color inferior a 100, apagará dicho canal; un porcentaje mayor a 100 se saturará; si vale 100, se devolverá su color original. Controlando los tres canales a la vez podemos aumentar o disminuir la luminosidad de la imagen. De esta forma, si disminuimos a la vez el porcentaje de RGB a 0 tendremos un fundido a negro y si aumentamos el porcentaje de RGB a 200, se realizará un fundido a blanco. Por tanto, el fundido de estas sentencias son equivalentes:

Fade_on() = fade(100, 100, 100, 1);
Fade_off() = fade (0, 0, 0, 1);

Además de estos tres parámetros de color existe un cuarto parámetro que controla la velocidad a la que se realiza un fundido. Un valor de 1 indicará un fundido lento, mientras que un valor de 10 indicará un fundido muy rápido. Si le damos como valor un número mayor de 64, tendremos un fundido instantáneo.

Podemos hacer fundidos de pantalla de diversas maneras, esto dará vistosidad a nuestros juegos

Es importante saber que el fundido no se realizará de forma instantánea, sino que se realizará en cada frame. Es por ello que, para controlar si el fundido ha llegado a su fin (por ejemplo si no deseamos que se destruyan los procesos hasta que el fundido haya finalizado), debemos utilizar la variable global predefinida *fading*, que vale "cierto" si el fundido se está realizando aún y "falso" si todavía se encuentra en proceso.

Además de estas funciones, podemos realizar otro efecto bastante llamativo, pero a la vez complicado, que es la rotación de paletas. Y decimos complicado no por el efecto en sí, que se realiza con una sola función, sino por el diseño de las imágenes y de la paleta que este efecto implica. Este efecto consiste, como bien indica su nombre, en realizar una rotación de los valores RGB de una paleta completa o bien de una parte de ella. De esta forma, un mismo índice contendrá un valor distinto cada vez. Por tanto debemos crear una imagen que contenga una serie de colores escalados cuyos índices sean consecutivos. Supongamos que utilizamos los colores de la paleta que van desde 32 a 47. Dibujamos una imagen cuadrada con estos colores consecutivamente (32,33,...,47). Si aplicamos el efecto de rotación a estos colores, provoca un efecto bastante llamativo. Veamos la sintaxis de esa función:

Roll_palette(<color inicial>, <número de colores>,<incremento>);

Donde "color inicial" indica el comienzo del rango de colores a rotar (en nuestro caso 32), "número de colores" indica el número de colores que contiene el rango (en nuestro caso 16) e "incremento" indica cuánto queremos desplazar los colores de la paleta. Un incremento de 1 realizará una rotación en un sentido, mientras que un valor de –1 realizará la rotación en el sentido opuesto. A mayor valor, mayor velocidad de rotación. Podemos ver, como claros ejemplos de utilización de estos efectos, los efectos de plasma y la barra inferior del logo de arranque de Windows 95.

Control mediante joystick

Si bien ya tratamos este tema con anterioridad, es mucho más amplio de lo que en un momento parecía. El acceso que realizamos en su momento era digital, es decir, o estaba girado el joystick hacia la derecha o no lo estaba, pero no podíamos medir la intensidad de este giro. Sin embargo, si realizamos un acceso analógico a dicho dispositivo, podemos sacar muchísimo más partido al control mediante el joystick. Este tipo de acceso consiste en obtener del joystick valores intermedios en los distintos ejes. De esta forma, si tenemos ligeramente girado el joystick hacia la derecha, el giro del objeto que controlemos será más lento que si tenemos el joystick girado completamente, caso en el cual girará mucho más rápido.

Además de esto, limitamos el uso de un sólo joystick. ¿Qué ocurre si tenemos pues conectados dos joysticks? Vamos a ver cómo podemos arreglar todo este embrollo.

Para simbolizar las posibles situaciones de una palanca de juegos (joystick analógico), debemos establecer un sistema de comunicación con el dispositivo. Mediante este sistema, el joystick nos devuelve un valor que, en nuestro caso, se encuentra aproximadamente entre 4 y 200. Un valor de 4 indicará un desplazamiento hacia la izquierda en el eje X, y un valor de 200 indicará un desplazamiento hacia la derecha. Por tanto, valores intermedios indicarán desplazamientos más leves, que traduciremos adecuadamente en nuestros programas. La función que utilizaremos para ello será la que sigue:

Get_joy_position (<nº del eje>);

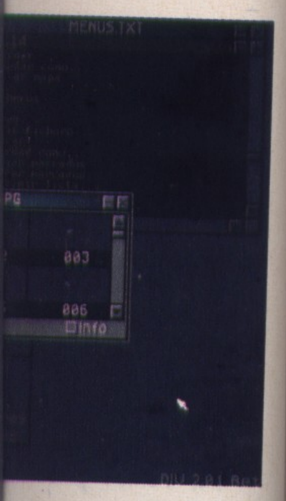
Siendo el número del eje el que sigue:

– 0 = eje X del joystick 1
– 1 = eje Y del joystick 1
– 2 = eje X del joystick 2
– 3 = eje Y del joystick 2

Devolviendo el valor adecuado del rango anteriormente establecido.

Como podemos observar a raíz de la tabla de posibles valores del número de ejes, podemos de esta forma obtener los datos del desplazamiento de un segundo joystick. Con ello disponemos ya de soporte de desplazamiento para dos joysticks. Además de los ejes, disponemos de los estados de los botones de ambos joysticks, como ya publicamos en el tercer artículo de esta serie, mediante la estructura joy. En esta estructura disponíamos de 4 botones cuyas correspondientes variables dentro de la estructura eran *button, button2, button3* y *button4*. Para dos joysticks tomaremos los valores de button3 y button4 como los botones 1 y 2 del segundo joystick.

Para mejorar los resultados de la lectura analógica y evitar la pre-



limitamos el
ick. ¿Qué ocu-
conectados
s a ver cómo
do este

as posibles
alanca de jue-
co), debemos
a de comuni-
tivo. Mediante
ck nos devuel-
nuestro caso,
nadamente
lor de 4 indi-
nto hacia la
y un valor de
lazamiento
tanto, valores
desplaza-
ue traducire-
en nuestros
n que utiliza-
a que sigue:

<nº del eje>);

del eje el que

stick 1
stick 1
stick 2
stick 2

or adecuado
te estableci-

bservar a raíz
valores del
mos de esta
os del despla-
do joystick.
a de soporte
a dos joys-
es, dispone-
los botones
mo ya publi-
culo de esta
ctura joy. En
amos de 4
ndientes
structura
utton3 y
cks tomare-
on3 y but-
1 y 2 del

ultados de
vitar la pre-

sencia de picos, es decir, valores no acordes con el desplazamiento real, se le suele aplicar un filtro al valor obtenido del joystick. Este filtro es un porcentaje de reducción que se le aplica al rango de valores. Esto nos permite tener desplazamientos más suaves, pero a costa de un ligero retardo en la recepción de los datos. El valor de este filtro en porcentaje se puede modificar accediendo a la variable global *joy_filter*, la cual tiene un valor por defecto de 10, pudiendo adoptar valores desde 0 hasta 99.

Además del control de joysticks analógicos, disponemos en DIV de un sistema avanzado de detección de los mismos. Para conocer con exactitud si se encuentra un joystick conectado a nuestro equipo, podemos utilizar la variable global del sistema *joy_status*. Esta variable puede tomar 3 valores bien distintos, que utilizándolos adecuadamente podremos aprovecharlos por ejemplo en un menú de opciones para dar a elegir los controles del juego al usuario de nuestro programa. Veamos sus posibles valores:

- 0: el sistema no ha encontrado ningún joystick conectado al puerto. También tomará este valor si el joystick ha sido desconectado durante la ejecución del programa.
- 1: este es el valor por defecto de esta variable. Si al comienzo del programa o durante la ejecución no se detecta el joystick, el sistema dará a esta variable el valor 0, es decir, considerará al joystick como desactivado.
- 2: con este valor, DIV considerará siempre como activado el joystick, aunque se desconecte dicho dispositivo. Esto puede provocar que el sistema se ralentice en demasía si no se encuentra conectado el joystick, sin embargo, ahorrará gran cantidad de comprobaciones que realiza internamente DIV de forma periódica para comprobar que el joystick se encuentra conectado.

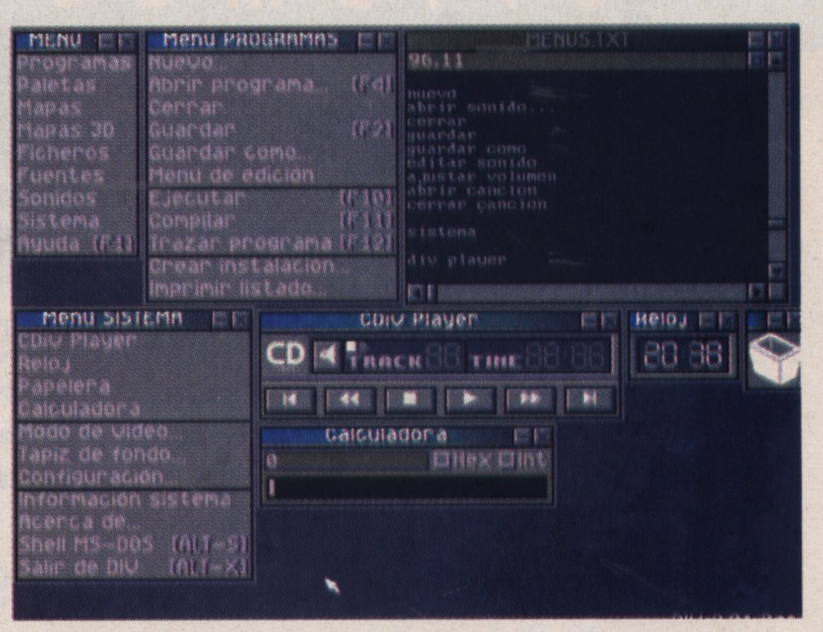
Por tanto, para detectar si tenemos un joystick conectado escribiremos:

```
[...]
IF (joy_status == 0)
    // deshabilitar las opciones de joystick
ELSE
    IF (joy_status == 1)
        // joystick conectado
    ELSE
        // joystick en estado activo permanente
    END
END
```

Además de este uso para la detección, podemos utilizar la variable *joys_status* para desactivar temporalmente o de forma permanente el uso del joystick. Por ejemplo, si termina una carrera de coches y queremos que no siga desplazándose al antojo del jugador el coche, podemos desactivarle el joystick dando el valor nulo a la variable del sistema, haciendo posteriormente que el coche frene sin ningún tipo de problema.

Cambiando la velocidad de nuestros programas

Para terminar, veamos cómo podemos cambiar la velocidad de ejecución de nuestros juegos manipulando el número de frames o imágenes por segundo que se mostrarán. Por defecto, el valor que tiene DIV es de 18 imágenes por segundo, que es bastante, si bien un valor de 24 imágenes dará una calidad superior y a la vez suficiente. Pero tenemos un problema en ordenadores menos potentes, y es que, si no se pueden mostrar ese número de imágenes por segundo, el sistema se verá seriamente ralentizado y por tanto el juego perderá probablemente gran parte de la jugabilidad. Sin embargo no ocurrirá al revés, y en sistemas más rápidos la velocidad se conserva constante. Es por ello que podemos dejar a DIV que se permita la licencia de no visualizar ciertos frames para con ello ganar velocidad, pero a costa de una menor calidad de imagen. Es por ello que utilizaremos la función *set_fps()* para cambiar estos valores. Veamos su sintaxis:



Realizando un acceso analógico al joystick podemos sacar muchísimo más partido de su control

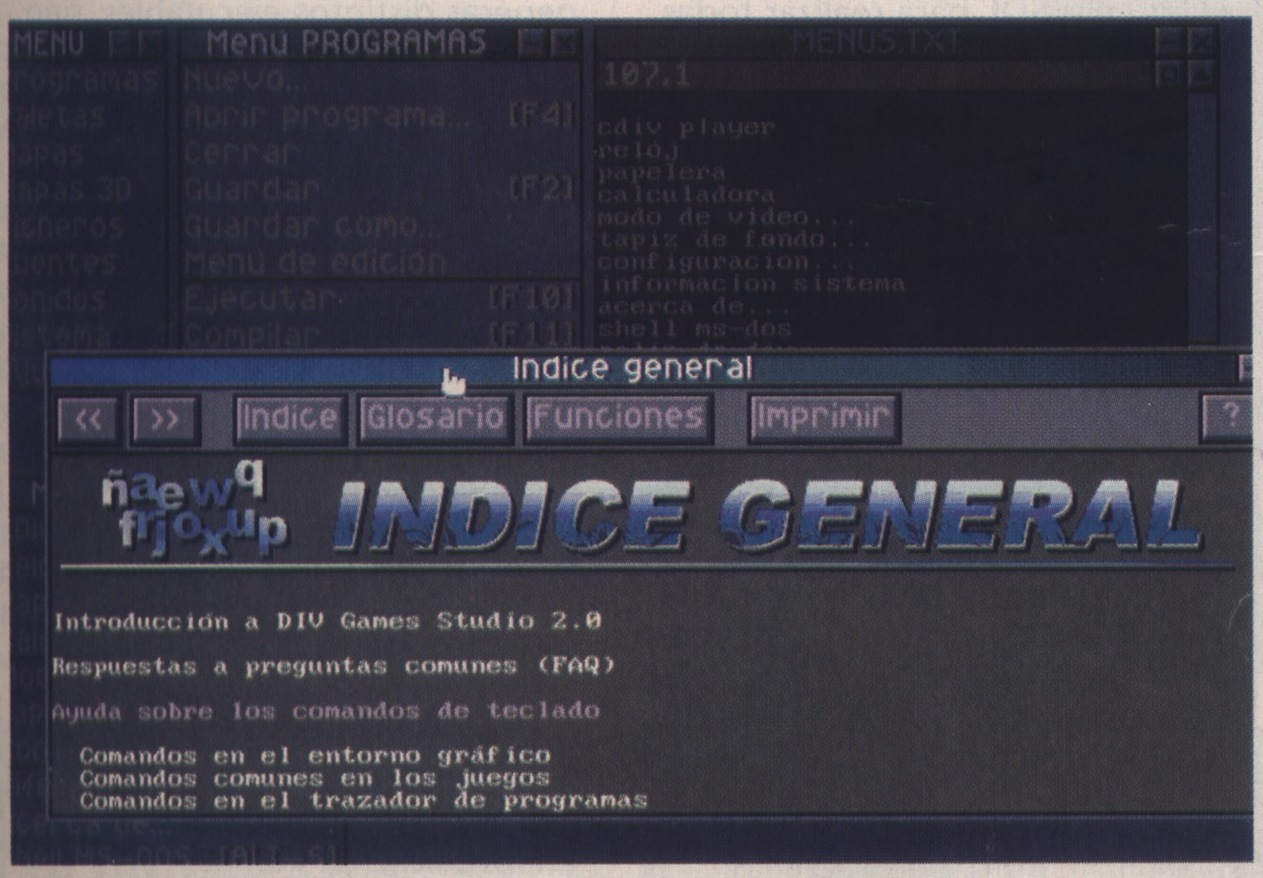
```
Set_fps (<imágenes por segundo>, <saltos permitidos>);
```

El primer parámetro indica las imágenes por segundo (por defecto 18) y el segundo el número de imágenes consecutivas que puede saltarse el programa durante la ejecución. Un valor de 1 puede bastar para obtener una relación calidad/velocidad más que suficiente en ordenadores lentos.

Conclusiones

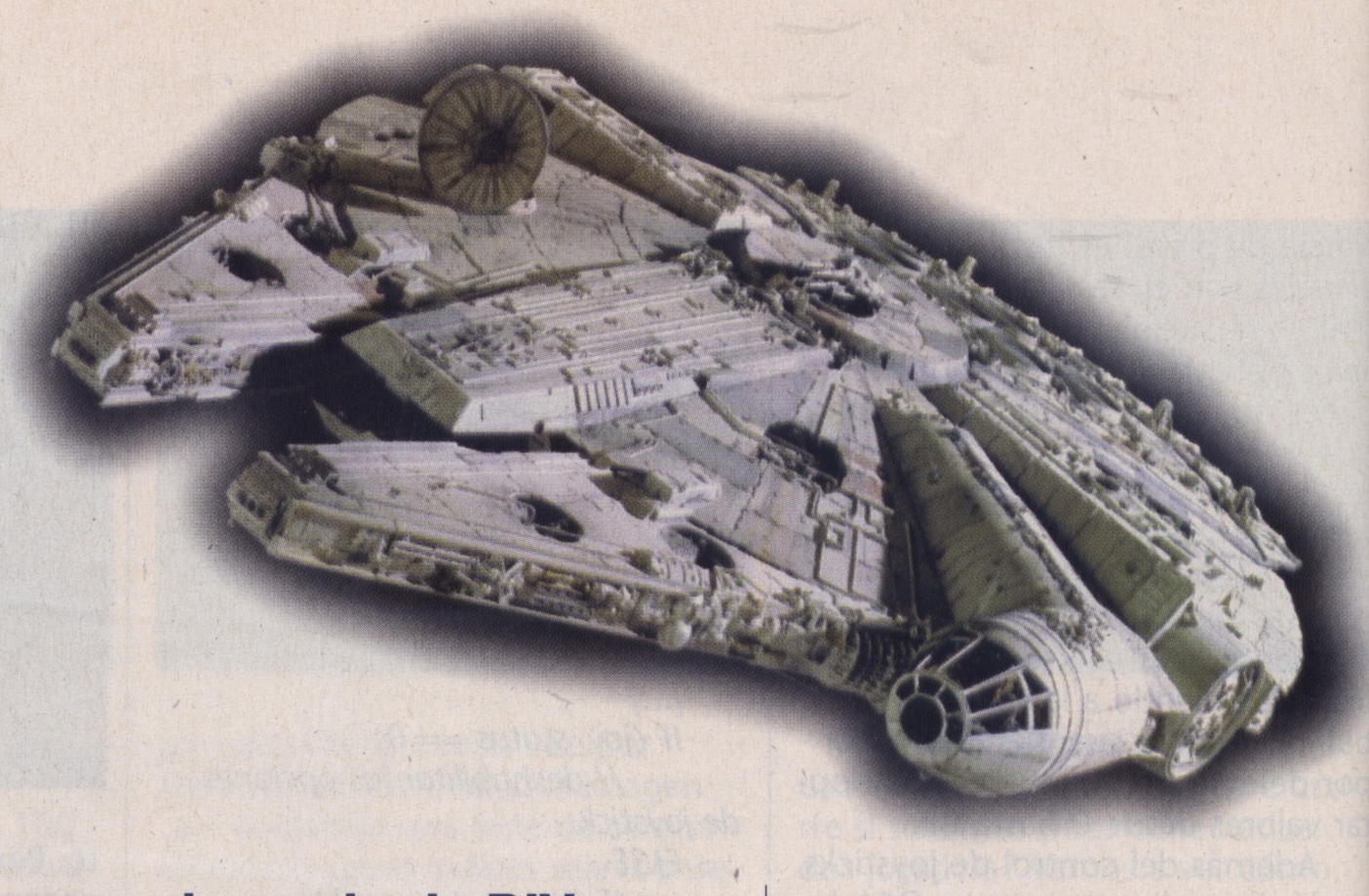
Con esta cantidad de datos esperamos que vuestras producciones mejoren, si no considerablemente un poco al menos, y de paso, le saquemos más jugo a DIV. Por este número ya es suficiente, como siempre practiquen. En próximos números nos gustaría realizar algún juego de demostración para comentarlo en todos sus pasos de desarrollo. Es por ello que necesitamos las sugerencias de los lectores. Estas pueden ser enviadas junto con cualquier tipo de pregunta, crítica, duda o cualquier cosa que en definitiva os pase por la cabeza a la dirección trinidad@arrakis.es.

Pablo Trinidad



Fénix

Un nuevo DIV



Hace ya algunos meses, el mundo de DIV se vio gratamente sorprendido por la aparición de un programa que revolucionó en varios aspectos el mundo de DIV: Fénix, una especie de DIV para Windows y Linux.

Casi desde la aparición de DIV, los usuarios han ansiado la puesta a punto de una versión que generase juegos que funcionasen en Windows 95/98 o bien en Linux. Dicha versión vino de la mano de José Luis Cebrián, que hizo pública la primera versión beta de este programa con el nombre de DIVC (DIV compiler). En poco tiempo se realizaron numerosas mejoras y muchos de los errores se corrigieron hasta conseguir una versión casi definitiva de lo que se llamaría al final FÉNIX. Veamos las características de este gran programa.

Fénix, un programa de libre distribución

Si por algo se caracteriza este programa es por su distribución totalmente gratuita bajo la licencia GNU GPL (GNU General Public License). Esto significa que cualquier persona puede utilizar este programa de forma gratuita, incluso realizar modificaciones de éste; sin embargo, en este último caso, la redistribución de la versión modificada debe realizarse también bajo los términos de esta licencia.

Gracias a esto, y a la publicación del código fuente del programa, rápidamente surgió una gran cantidad de adeptos que deseaban ayudar en lo posible en el futuro desarrollo del programa.

Un programa multiplataforma

La gran característica de este programa, y lo que realmente llamó la atención fue el funcionamiento de Fénix en plataformas como Linux y Windows 95/98. Si bien José Luis Cebrián desarrolló su programa en el entorno Linux, su incansable trabajo y la colaboración de numerosos DIVeros/as, llevó a realizar una versión para Windows 95/98. Esto se debió en gran parte al uso de las librerías SDL para realizar todas las operaciones de visualización, librerías que se caracterizan por su gratuidad y desarrollo multiplataforma.

Pero, ¿qué es exactamente Fénix?

Aunque muchos crean que se trata de un simple compilador de DIV que funciona en otros sistemas, Fénix va mucho más allá, si bien parte de ello es cierto. Permite ejecutar casi todos los programas de DIV, y decimos *casi* porque al ser una versión beta, es decir, aún en fase de desarrollo, quedan aún algunos flecos por cubrir. Pero no sólo eso, Fénix añade tal cantidad de novedades al lenguaje que podemos considerarlo casi como un compilador distinto de DIV. Funciones de manipulación de ficheros, tipos de datos y conversiones de tipos entre otros, son algunas de las nuevas incorporaciones al lenguaje.

¿Cómo funciona Fénix?

Fénix, al igual que DIV, es un lenguaje pseudo-interpretado. Pero, ¿qué quiere decir eso exactamente? El concepto es el mismo que se da, por ejemplo, en el lenguaje Java. Para que se pueda ejecutar el código, el compilador no genera código autoejecutable (lo que comúnmente se conoce como archivo EXE), sino que genera un código llamado *bytecode* que posteriormente podrá ejecutar un intérprete. De esta forma, si queremos que un programa funcione en varias plataformas, no necesitamos generar distintos ejecutables, sino que sólo debemos generar un sólo archivo con el código *bytecode* que entiende el intérprete y ejecutarlo en un intérprete específico de la plataforma deseada. Esta fue una de las características que le dio a Java tanto éxito y que, sin duda, también lo ha hecho en este caso.

Pero ¿cómo influye esto a la hora de la utilización de Fénix? Este programa incluye dos ejecutables que nos permitirán ejecutar nuestros programas de DIV. Estos son FXC (compilador) y FXI (intérprete). Para ejecutar cualquier programa PRG de DIV, debemos utilizar la siguiente sintaxis:

FXC fichero.PRG

***Galax* fue el primer juego de DIV que funcionó perfectamente en Fénix.**

DIV developer

NÚMERO 7

Sumario

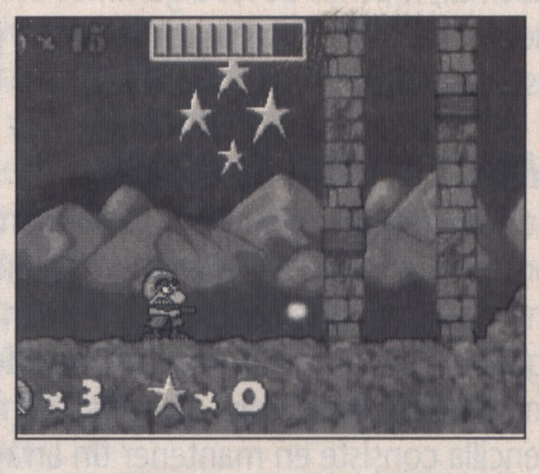
Curso de programación y concurso

Continuamos con nuestros cursos de programación habituales siguiendo los capítulos ya iniciados. Como siempre, esperamos que os sean de utilidad.
En cuanto a nuestro concurso, por supuesto sigue en pie con los mismos premios. Ya sabéis: 25.000 pesetas para el ganador y 20.000 para los otros dos juegos elegidos. Esperamos con impaciencia recibir vuestros juegos.
Aquí tenéis los ganadores de este número. Esperamos que os gusten.

Vuestras sugerencias

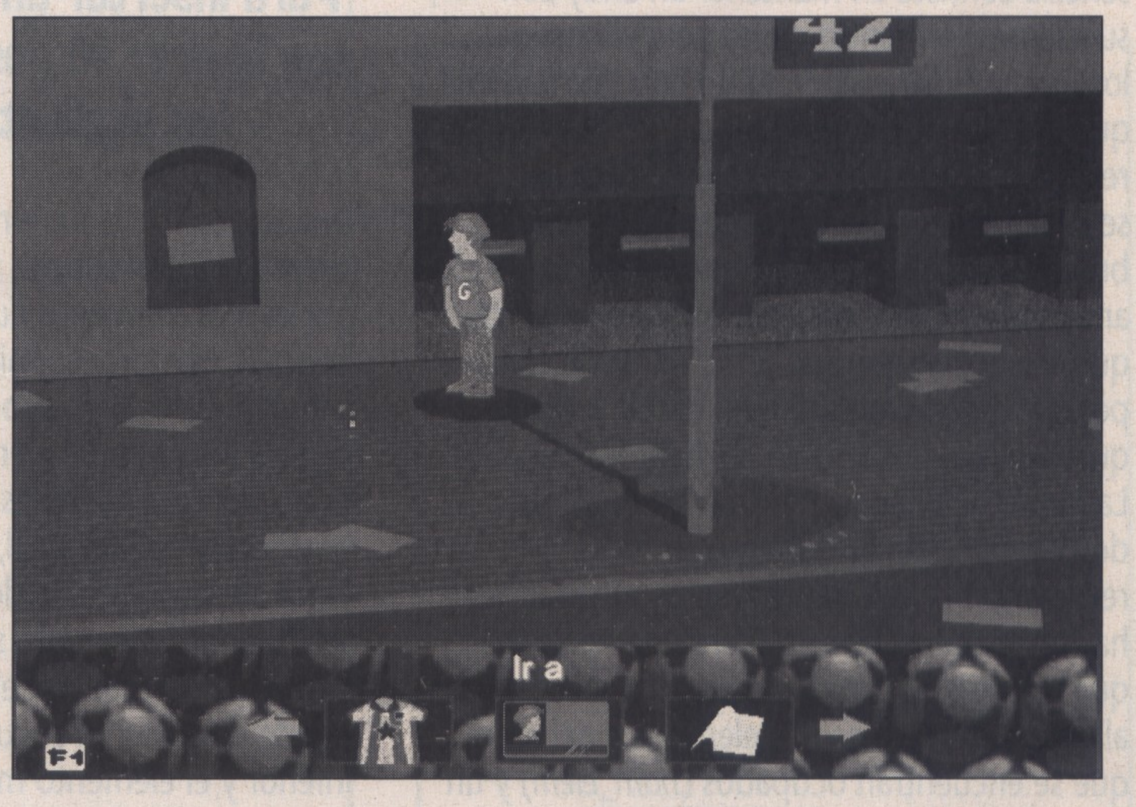
Queremos haceros saber que todas las ideas, críticas o sugerencias serán bienvenidas por parte de esta redacción. Si tenéis inquietud por algún tema que no tocamos demasiado, si consideráis que alguna sección de esta revista debe desaparecer o debe ser cambiada por otra, incluso si os animáis a proponernos artículos realizados por vosotros/as, que creéis pueden ser aprovechados por otros usuarios/as, no dudéis en poneros en contacto con nosotros. Os necesitamos para mejorar.
Para mandarnos los juegos o cualquier sugerencia sobre la publicación, podéis hacerlo por e-mail (**gover@prensatecnica.com**), correo normal o incluso entregarlos a mano en la dirección de la redacción:

Divmanía
C/ Alfonso Gómez, 42
Nave 1-1-2
28037, Madrid, España



Para los juegos del concurso

No os olvidéis meter vuestros datos completos cuando mandéis juegos al concurso. Lo decimos porque hemos recibido algunos en los que falta la ciudad, el teléfono o directamente la dirección entera.
Es conveniente que mandéis vuestro nombre completo, vuestra dirección (con ciudad y código postal), teléfono y dirección de correo electrónico, si la tenéis.
Muchas veces nos llegan juegos con problemas y no podemos ponernos en contacto con vosotros. Por ejemplo, enviamos un mensaje a Rubén Melo Delgado que nos ha enviado un juego que no contiene ningún ejecutable, y a Ariel Antonio Alvarez García, cuyo juego da un error. Si leéis este mensaje poneros en contacto con la redacción de DIVmanía. Recordaros el teléfono: 91 304 06 22.
También recordaros que es vital que introduzcáis un fichero de texto que contenga la presentación de la historia, vuestras fuentes de inspiración, las características del juego, su instalación y, sobre todo, el código fuente.



Destacamos

En nuestro CD incluimos como siempre los juegos que han resultado ganadores en este número y, cómo no, sus respectivos códigos para que veáis todo el proceso de creación. Pero además, en este número tenemos jugosos programas para vosotros. Fénix, Poser 2 y Bryce 2, en versiones completas, y la demo de Bryce 4, además, como siempre, de los archivos que complementan los artículos de la revista.

PROGRAMACIÓN BÁSICA

Métodos de ordenación

El objetivo principal de este artículo es presentar una serie de algoritmos que muestren cómo ordenar un conjunto de elementos. La variedad de estos métodos servirá para mostrar cómo una tarea puede realizarse de diferentes formas, junto con sus ventajas e inconvenientes.

En este artículo se presentarán algunos de los algoritmos más utilizados para ordenar una serie de elementos almacenados en un array. Los procesos de ordenación permiten realizar búsquedas más eficientes de un elemento dentro de una serie ordenada, tarea muy habitual en la programación. La descripción de varios métodos de ordenación permitirá al lector seleccionar el más adecuado en cada momento, pues no siempre el más eficiente respecto al tiempo de ejecución, suele ser el más adecuado (por ejemplo, si hay restricciones de memoria). Además, muchos de los algoritmos utilizados al desarrollar programas se construyen a partir de estos algoritmos.

MANTENIENDO EL ORDEN

Supóngase que el número de elementos a ordenar es variable y que éstos deben ser insertados en un conjunto de elementos de iguales características, que a su vez deben mantenerse ordenados. La solución más sencilla consiste en mantener un array con suficientes elementos como para almacenar a los que se espera tener, junto con una variable que indique el número de posiciones realmente ocupadas en el array. Cada vez que se desee insertar un elemento, basta con buscar la posición que éste debe ocupar en el array y desplazar una posición los elementos que se encuentren situados a partir de dicha posición, incrementando en uno la variable que indica el número de elementos ocupados. La figura 1 contiene el procedimiento que describe el anterior tratamiento. Para representar la lista de elementos ordenados se ha construido el tipo *lista_ord* que no es más que un registro constituido por un campo para almacenar el número de elementos del array que se encuentran ocupados (*num_elem*) y un array de cadenas de caracteres, donde se almacenan los elementos que deben estar ordenados (*arr_cadena*). El procedimiento *inserta_elemento* dispone de dos parámetros, el elemento a insertar y la lista donde se insertará; el procedimiento comienza buscando el lugar que debe ocupar el elemento insertado, una vez que se ha encontrado, los elementos situados en posiciones superiores se trasladan a las siguientes, para dejar su lugar al elemento a insertar, posteriormente, se incrementa el número de elementos de la lista. Este algoritmo es muy ineficaz, pues cada vez que se inserta un elemento en la lista es necesario mover todos los elementos situados entre la posición que se ha localizado y el último elemento ocupado en la lista, además, la búsqueda de la posición donde se insertará el elemento es secuencial. El número medio de movimientos que se realizan para insertar un nuevo elemento es la mitad de los elementos que contiene el array, al igual ocurre con el número medio de comparaciones que han de realizarse hasta que se encuentra el lugar que deberá ocupar el elemento a insertar.

Para insertar un elemento, basta con buscar su posición y desplazar los siguientes un lugar

Una mejora del anterior algoritmo lo constituye la búsqueda binaria, representada por el procedimiento *busca_biparticion* de la figura 2. Este procedimiento mantiene en dos variables de tipo entero los límites donde se deberá encontrar el elemento buscado (al principio estas variables contienen las posiciones inicial y final, respectivamente), a partir de los límites, y de forma iterativa, se selecciona el elemento situado en la mitad; comparando éste con el elemento a buscar se puede decidir si se encuentra entre el elemento inferior y el elemento mitad o entre el elemento mitad y el elemento superior, siendo éstos (según el caso) los nuevos límites inferior y superior, respectivamente, para continuar la búsqueda. Sin duda, este algoritmo de búsqueda es mejor que su homólogo secuencial, pues en cada iteración se eliminan del espacio de búsqueda la mitad de los elementos originales en los que se debe buscar. Con unas pocas modificaciones del procedimiento *busca_particion* es posible dotar al procedimiento *inserta_elemento* de búsqueda binaria y así hacerlo más eficiente.
Otro factor que hace ineficaz al algoritmo inicial de inserción en un array ordenado lo constituye el desplazamiento de los elementos del array para poder acomodar al nuevo elemento. Este problema es debido a que la posición que ocupa un elemento en el array es a su vez la que establece el orden de los elementos. Es posible resolver este problema implantando una lista enlazada que tenga como base el array.
En la figura 3 se muestra la estructura de datos para implementar una lista enlazada, en el tipo de datos *lista* se puede encontrar entre otros componentes un array donde se almacenarán los datos a mantener ordenados, los elementos de este array son registros que se componen a su vez de un campo para almacenar la información y otro campo que contendrá la posición del siguiente elemento dentro del array, de esta forma, el orden de los elementos no se corresponde con el orden natural del array. Además, el tipo de datos *lista* contiene dos campos de un tipo enumerado: el campo *principio_l* contiene la posición del array donde se encuentra el primer elemento de la lista, el campo *libre_l* contiene la primera posición del array no ocupada; este campo es necesario pues los elementos libres del array ya no se encontrarán al final de éste, sino que están enlazados en una lista de elementos libres.
El procedimiento *inicia_lista* de la figura 3 muestra cómo antes de utilizar una variable de

```
procedure inserta_elemento (var l: lista_ord; c: cadena);
  var
   i, j: integer;
  begin
   if lista_vacia(l)
     then
      with l
        do
         begin
          num_elem:= 1;
          arr_cadena[num_elem]:= c;
         end
      else
       if l.num_elem = max_elem
         then
          writeln('La lista está llena ')
         else
          begin
           i:= 1;
           while (l.arr_cadena[i] < c) and (i <= numero_elementos(l) )
             do
              i:= i + 1;
           for j:= l.num_elem downto i
            do
             l.arr_cadena[j+1]:= l.arr_cadena[j];
           l.arr_cadena[i]:= c;
           l.num_elem:= l.num_elem + 1
          end;
  end;

Estructuras de datos

const
 max_elem= 30;
type
 cadena= string[50];
 lista_pal= array [1..max_elem] of cadena;
 lista_ord= record
              arr_cadena:lista_pal;
              num_elem: integer;
            end;
```

FIGURA 1.

tipo *lista* hay que tratarla de tal forma que se indique que todos sus elementos están libres. Así, este procedimiento asigna al principio de la lista el valor 0 (representado por la constante *final*), indicando de esta forma que la lista está vacía y el valor para los elementos libres se establece a 1 (primer elemento libre). Posteriormente cada uno de los elementos del array se enlazan con el siguiente para formar una lista de elementos libres (este tratamiento está representado por el conjunto de instrucciones de la instrucción *for*), el último elemento de dicha lista contendrá en su campo *sig* el valor 0 (representado como la constante *final*) pues dicho elemento no tiene elemento siguiente.
La figura 4 contiene un procedimiento para almacenar de forma ordenada un nuevo elemento. Este procedimiento comienza comprobando si hay espacio libre en la lista, esto es, si el campo *libre_l* contiene un valor distinto de cero, si hay espacio libre entonces se procede a comprobar si la lista está vacía, en cuyo caso será el campo *principio_l* el que se encontrará con valor igual a cero. Si es así, se selecciona el primer elemento libre y se introduce como primer elemento de la lista. Si la lista no se encontraba vacía, entonces, se procede a comparar el primer elemento de la lista, es decir, aquel cuya posición es la indicada por el campo *principio_l*, si dicho elemento es mayor, la inserción del elemento se realizará al principio de la lista, de nuevo se toma una posición libre del array, y el campo *sig* de dicho elemento pasará a ser el valor del campo *principio_l* y puesto que el principio de la lista ha sido modificado, el campo *principio_l* pasa a contener la posición del elemento que se seleccionó anteriormente.

Si ninguno de los anteriores casos se da, entonces, se procede a la búsqueda de los elementos entre los que se debe insertar el nuevo, esta función es la que tienen asignadas las variables *actual* y *adelante*. La variable *adelante* siempre apunta al siguiente elemento de la lista al que apunta el elemento *actual*, cada vez que se comprueba que el elemento apuntado por la variable *adelante* es menor que el elemento a insertar las anteriores variables pasan a apuntar a sus respectivos elementos siguientes. El anterior proceso se detendrá cuando el elemento al que apunta la variable *adelante* sea mayor que el elemento a insertar o bien se llegue al final de la lista, finalizando el procedimiento con la modificación de los correspondientes campos *sig* de los elementos del array implicados en la inserción.

Adelante siempre apunta al siguiente elemento de la lista que apunta al elemento _actual_

El uso de la búsqueda binaria no es posible utilizarlo en este algoritmo pues el orden no se establece mediante las posiciones del array, sino por el campo *sig*, aunque sí es cierto que este algoritmo evita tener que desplazar los elementos de un array cada vez que se realice una inserción.

METODOS DE ORDENACION

Todos los algoritmos descritos en este apartado supondrán que los elementos que hay que ordenar se encuentran almacenados en un array, a diferencia de los anteriores que permitían la inserción de un elemento en un conjunto previamente ordenado.
El primer método de ordenación que se describirá será el método de inserción directa. Éste consiste en mantener ordenada una partición del array (inicialmente ésta solo constará del primer elemento); en cada iteración esta partición se aumentará en un elemento hasta llegar al elemento final del array. Es evidente que para mantener ordenada cada una de las particiones correspondiente a cada una de las iteraciones del algoritmo, habrá que trasladar el elemento que se ha introducido a la posición que le corresponde, moviendo con dirección hacia el final del array aquellos elementos que estando ordenados con anterioridad sean mayores que el anterior elemento. La terminación del proceso de traslación, cuando se está tratando un elemento, se producirá al encontrar un elemento que sea menor que el que está siendo tratado o bien cuando se haya llegado al principio del array (no hay más elementos que comparar).

```
const
  max_elem= 100; (número de elementos disponibles para la lista)
  final= 0; (final de la lista)
  long_cad= 50;(longitud de una cadena de caracteres)
type
  siguiente= 0..max_elem;
  cadena= string[long_cad];
  elemento= record
              contenido: cadena;
              sig: siguiente;
            end;
  lista_elem= array[1..max_elem] of elemento;
  lista= record
           almacen: lista_elem;
           principio_l: siguiente;
           libre_l: siguiente;
         end;
procedure inicia_lista (var l: lista);
  var
   i: siguiente;
  begin
   with l
     do
      begin
       principio_l:= 0;
       libre_l:= 1;
       for i:= 1 to (max_elem -1)
         do almacen[i].sig:= i + 1;
       almacen[max_elem].sig:= final; (el último elemento apunta al final)
      end;
end;
```

FIGURA 3.

```
procedure busca_biparticion (l: lista_ord; c: cadena; var b: boolean;
                             var posicion: integer);
  var
   sup, inf, mitad: integer;
  begin
   b:= false;
   sup:= numero_elementos (l);
   inf:= 1;
   repeat
    mitad:= (inf + sup) div 2;
    if c > l.arr_cadena[mitad]
      then
       inf:= mitad + 1
      else
       sup:= mitad - 1;
   until (c = l.arr_cadena[mitad]) or (sup < inf);
   if l.arr_cadena[mitad] = c
     then
      begin
       b:= true;
       posicion:= mitad
      end
     else
       posicion:= 0;
end;
```

FIGURA 2.

```
procedure inserta_en_orden (cad: cadena; var l:lista);
  var
   actual, aux, adelante: siguiente;
  begin
   if hay_espacio (l)
     then
      if lista vacia (l)
        then
         with l
           do begin
              principio_l:= libre_l;
              libre_l:= almacen[libre_l].sig;
              almacen[principio_l].contenido:= cad;
              almacen[principio_l].sig:= final
             end
        else
         if (l.almacen[l.principio_l].contenido > cad)
           then {insercion al principio de la lista}
            with l
              do begin
                 aux:= libre_l;
                 libre_l:= almacen[libre_l].sig;
                 almacen[aux].sig:= principio_l;
                 almacen[aux].contenido:= cad;
                 principio_l:= aux;
                end
           else
            with l
              do begin
                 actual:= principio_l;
                 adelante:= almacen[actual].sig;
                 while(adelante <> final )and (almacen[adelante].contenido<cad)
                   do begin
                      actual:= adelante;
                      adelante:= almacen[adelante].sig;
                     end;
                 aux:= libre_l;
                 libre_l:= almacen[libre_l].sig;
                 almacen[aux].contenido:= cad;
                 almacen[actual].sig:= aux;
                 almacen[aux].sig:= adelante;
                end
   else writeln ('ERROR: NO HAY ESPACIO PARA INSERTAR UN NUEVO ELEMENTO');
end;
```

FIGURA 4.

PROGRAMACIÓN EN C

Variables externas, directiva y macros

Este capítulo se va a dedicar a explicar tres temas muy importantes: las variables externas o globales, que permiten ser usadas en cualquier punto del programa, las directivas, esenciales para realizar algunas operaciones básicas, y las macros, elementos muy útiles para el programa.

Las variables que hemos empleado en los programas que se dieron de ejemplo en capítulos anteriores estaban confinadas a las funciones que las utilizaban, es decir, eran *visibles* o accesibles únicamente para las funciones donde se declaraban. Las variables que se declaran dentro de una función particular y que son utilizadas solamente por ellas, se denominan *locales* (o automáticas). En la mayoría de los casos se prefiere utilizar este tipo de variables, pero a veces es deseable emplear otras que no sean conocidas por todas las funciones de un programa. Esto es válido, por ejemplo, en el caso de que todas las funciones deban leer o modificar una variable determinada, siendo incómodo y, a veces, impracticable, comunicar su valor de una función a otra mediante el uso de argumentos y valores de retorno. Veamos un ejemplo que utiliza una variable externa:

```
/* externa.c   */
/* comprueba el uso de las variables externas */
void parimpar(void);
void negativo(void);

int numclave;

main( )
 {
  printf("Escriba numclave: ");
  scanf("%d", numclave);
  parimpar( );
  negativo( );
 }

/* parimpar */
/* chequea si numclave es par o impar */
void parimpar(void)
 {
  if(numclave % 2)
      printf("Numclave es impar.\n");
    else
      printf("Numclave es par.\n);
 }

/* negativo */
/* comprueba si numclave es negativo */
void negativo(void)
 {

  if(numclave<0)
      printf("Numclave es negativa.\n");
    else
      printf("Numclave es positiva.\n");
  }
```

Las directivas constituyen un lenguaje dentro del propio C

En este programa, las funciones *main( )*, *parimpar( )* y *negativo( )* tienen acceso a la variable *numclave*. Para que esta variable tenga carácter global hay que declararla fuera de las funciones, incluida *main( )*. Por ello aparece antes de la definición de *main*.
Como se puede ver, ejecutando el programa *main( )* da un valor a la variable *numclave* y las funciones *parimpar( )* y *negativo( )* pueden leer ese valor.

Hay más comentarios que añadir sobre la visibilidad de las variables y sobre otros temas relacionados con su tiempo de duración (tiempo de vida).
Estas cuestiones están emparejadas con otro asunto: la capacidad que tiene el C de combinar en un único programa ejecutable, durante el enlazado, ficheros compilados de forma separada. Volveremos a hablar de este tema en un capitulo más avanzado.
Sin embargo, tenemos que advertir de los peligros que conlleva el uso indiscriminado de variables externas. Puede parecer tentador simplificar las cosas definiendo como externas todas las variables; los programadores de BASIC tienden a ser víctimas de esta práctica. Sin embargo, existen varias razones para que ésta no sea una buena idea. Primero, las variables externas no están protegidas de alteraciones accidentales realizadas por funciones que, en principio, no están programadas para modificarlas. Segundo, y como veremos en otro capítulo más avanzado, las variables externas utilizan la memoria de una manera menos eficiente que las variables locales. La regla de oro es: las variables deben ser locales a menos que haya una muy buena razón para hacerlas externas.

DIRECTIVAS DEL PREPROCESADOR

En este instante cambiamos de ruta y vamos a ver la utilización de las directivas del preprocesador, que constituyen un lenguaje dentro del lenguaje C. Esta capacidad no existe en otros lenguajes de alto nivel (aunque sí en ensamblador).
Para entender las directivas del preprocesador, veamos primero qué hace un compilador. Cuando escribe una línea de código:

```
num = 44;
```

le está pidiendo al compilador que traduzca este código a lenguaje máquina que luego será ejecutado por el microprocesador del ordenador. De esta forma, la mayor parte de

sus listados son instrucciones al microprocesador. Las directivas del preprocesador, por el contrario, son instrucciones al propio compilador. En lugar de ser traducidas a lenguaje máquina, el compilador las maneja directamente antes de que comience el proceso de compilación; de ahí el nombre de preprocesador.
Las directivas del preprocesador empiezan siempre con el símbolo #. Se pueden colocar en cualquier sitio del programa, pero frecuentemente van al principio del fichero, antes que *main( )*, o delante del comienzo de las otras funciones.

LA DIRECTIVA '#DEFINE'

El cometido más simple de la directiva *#define* es asignar un nombre (tal como DIAS_ANYO o PI) a constantes (tales como 385 o 3.14). Como ejemplo, veamos una modificación del programa *esfera.c* visto anteriormente en este capítulo. Veamos el mismo programa del anterior capítulo modificado:

```
/* esfera2.c */
/* calcula la superficie de una esfera */
#define PI 3.14159
float area(float);

main( )
{
 float radio;

 printf("Introduzca el radio de la esfera: ");
 scanf("%f, &radio);
 printf("El área de la esfera es %.2f, area(radio) );
}

/* area */
/* calcula el area de la esfera */
```

#define sirve para asignar un nombre a una constante

```
float area(float rad)
{
 return(4* PI * rad * rad);
}
```

En esta nueva versión del programa, el preprocesador busca primero todas las líneas del programa que comiencen con el signo #. Una vez encontrada la directiva *#define*, recorre completamente el programa, y donde encuentre PI coloca la expresión 3.14159. Éste es un proceso mecánico: simplemente se sustituye un grupo de caracteres, "PI", por otro "3.14159". Se parece bastante a una "búsqueda y sustitución global" en un procesador de textos.

Ejemplo:

```
#define PI 3.14159
```

La expresión situada inmediatamente detrás de *#define* (PI), que se buscará en todo el listado, se denomina *identificador*. La expresión que se encuentra a su derecha, que será la que reemplace a PI, se denomina *texto*. El identificador está separado del texto mediante un espacio. Por convenio, el identificador (en este caso PI) se escribe en mayúsculas. Esto facilita al lector la búsqueda de aquellas partes del programa que serán modificadas por la directiva *#define*.

¿POR QUÉ UTILIZAR '#DEFINE'?

Quizás se pregunte qué ha obtenido sustituyendo PI por 3.14159 en nuestro programa. Por una parte, hemos aumentado la legibilidad del programa. Aunque 3.14.159 es una constante tan común que es fácilmente reconocible, habrá ciertos casos en que una constante no revele tan claramente su propósito. Por ejemplo, como veremos más adelante, la expresión *\x1B[C* mueve el cursor un espacio hacia la derecha. Pero, ¿qué expresión le parece más seria de comprender si se la encuentra en medio de un programa *\1B[C* o *CURSOR_DERECHA*? Ésta es una de las razones para utilizar la directiva *#define*.

```
#define CURSOR_DERECHA "x1B[C"
```

Antes de que comience la compilación se reemplazará automáticamente la cadena *CURSOR_DERECHA*, en cualquier lugar donde aparezca, por *\x1B[C*.
Existe otra razón más importante para utilizar la directiva *#define*. Suponga que una constante como 3.14159 aparece repetidas veces en el programa. Suponga además que decide aumentar la precisión añadiendo otro dígito, necesita cambiar todas las ocurrencias 3.14159 por 3.14.1592. Normalmente tendría que recorrer todo el programa y cambiar manualmente todas las apariciones de la constante. Sin embargo, si usted ha definido 3.14159 como PI en una directiva *#define*, sólo necesitará hacer el cambio en la misma directiva del preprocesador

```
#define PI 3.141592.
```

El cambio se efectuará automáticamente en todas las apariciones de PI antes de que comience la compilación.

¿POR QUÉ NO UNA VARIABLE?

Una variable puede proporcionar un nombre mnemotécnico a una constante, además de permitir cambiar rápidamente el valor de la constante en todo el listado. Sin embargo, existen al menos tres razones para que esto no sea una buena idea. Primero, es poco eficaz, ya que el compilador genera un código más rápido y compacto para las constantes que para las variables. Segundo, utilizar una variable para almacenar un valor que realmente es una constante es poco original y dificulta la comprensión del programa. Si algo no cambia nunca, es confuso definirlo como una variable. Tercero, existe siempre el peligro de que se altere inadvertidamente el valor de la variable durante la ejecución del programa, por lo que no es tan *constante* como uno puede creer.

MACROS

La directiva *#define* es realmente bastante más poderosa que lo que hemos mostrado hasta ahora. Este poder adicional proviene de la capacidad que tiene *#define* de utilizar argumentos. Antes de abordar este problema, veamos otro ejemplo de la directiva *#define* sin argumentos, cuyo objeto es realizar una transición más clara. En este ejemplo se muestra que se puede utilizar la directiva *#define* no sólo para sustituir constantes, sino para reemplazar la instrucción que queramos. Suponga que su programa necesita imprimir un mensaje de "Error" en varios lugares del programa. Utilizaría la directiva:

```
#define ERROR printf("\nError.\n");
```

Entonces, si el programa tiene la siguiente instrucción:

```
if(entrada>640) ERROR
```

ésta será expandida por el preprocesador a:

```
if(entrada > 640) printf("\nError.\n");
```

Veamos ahora un ejemplo de *#define* con su argumento. Si nunca entendió demasiado bien el mecanismo de la función *printf( )* tendrá un montón de dudas incluso para imprimir un número, considere la siguiente alternativa:

```
/* macroprn.c */
/* demostración de las macros, con la instrucción
printf( ) */
#define PR(n) printf("%.2f\n",n);
main( )
{
 float num1 = 27.25;
 float num2;

 num2 = 1.0 / 3.0
 PR(num1);
 PR(num2);
}
```

CURSO DE ENSAMLADOR

Operaciones sobre bits

Se puede decir que saber usar las intrucciones de manejo de bits es fundamental para todo aquel que tenga la intención de crear algoritmos bajo ensamablador más o menos complejos. Con este artículo te daremos las pistas necesarias para lograrlo.

Si tiene un conjunto de datos similares, puede encontrar incómodo asignar a cada dato un nombre de variable. Por ejemplo, suponga que quiere calcular la temperatura media de una semana cualquiera. Si se asigna a la temperatura de cada día un nombre de variable, tendría que leer cada valor de forma separada:

```
printf("Introduzca la temperatura del domingo: ");
scanf("%d", &tempdom);
printf("Introduzca la temperatura del lunes:");
scanf("%d", &templun);
```

y además necesitaríamos una expresión como la siguiente para calcular el valor medio:

```
(tempdom + templun + tempmar + tempmier + tempjuev + tempvier + tempsab) / 7.
```

Como se ve, es verdaderamente un mal negocio, especialmente si quiere calcular la media de las temperaturas de un mes o de un año.
Por tanto necesitamos una forma adecuada para referirnos a tales conjuntos de datos de similares características. El *array* es la solución. Proporciona una forma de referirse a los elementos individuales de un conjunto, utilizando el mismo nombre de variable para todos los elementos, pero empleando distintos índices o números. Veamos cómo podemos resolver el problema del cálculo de la temperatura media semanal utilizando *arrays*:

```
/* temp.c */
/* media semanal de las temperaturas */
main( )
 {
  int temper[7];     /* declaración del array*/
  int dia, suma;

  for(dia= 0; dia<7; dia++)     /*pone temperaturas en array */
     {
       printf("Introduzca la temperatura para el día %d: ",dia);
       scanf("%d", &temper[dia]);
     }

  suma = 0;
```

Estas instrucciones son muy importantes para el desarrollo de algoritmos complejos

```
  for(día=0; dia<a7; dia++)
      suma+= temper[dia];
   printf("La media es %d.", suma/7);
}
```

Este programa lee siete temperaturas, las almacena en un *array*, y luego calcula la temperatura media, leyendo los valores almacenados en el *array*, sumándolos y dividiendo el resultado por 7.

DECLARACIÓN DE 'ARRAYS'

Un *array* es una colección de variables de un cierto tipo, colocadas de forma contigua en memoria. Como le pasa a otro tipo de variables, los *arrays* tienen que ser declarados, de esta forma el compilador conocerá el tipo y el tamaño que queremos para el array. En el siguiente ejemplo se muestra una declaración típica:

```
int temper[7];
```

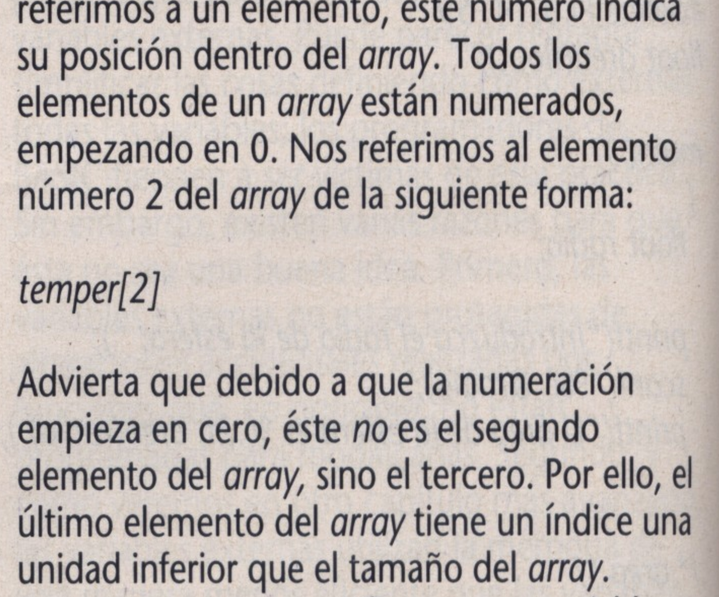
En este caso, *int* especifica el tipo de *array*, de la misma forma que en el caso de variables simples, y la palabra *temper* es su nombre. Sin embargo, el [7] es nuevo. Este número especifica cuántas variables de tipo *int* tendrá nuestro *array* (cada una de las variables en el *array* se denomina *elemento)*. Los corchetes le dicen al compilador que estamos definiendo un *array*.

REFERENCIAS A LOS ELEMENTOS INDIVIDUALES DE UN *ARRAY*

Una vez definido un *array*, necesitamos un sistema para referirnos a sus elementos individuales. Esto se realiza mediante índices, números entre corchetes que siguen al nombre del *array*. Sin embargo, hay que advertir que este número tiene dos significados diferentes: uno cuando se *refiere* a un elemento del *array* y otro cuando se *declara el array*, en este último caso el número encerrado entre corchetes especifica el tamaño del *array*. Cuando nos referimos a un elemento, este número indica su posición dentro del *array*. Todos los elementos de un *array* están numerados, empezando en 0. Nos referimos al elemento número 2 del *array* de la siguiente forma:

```
temper[2]
```

Advierta que debido a que la numeración empieza en cero, éste *no* es el segundo elemento del *array*, sino el tercero. Por ello, el último elemento del *array* tiene un índice una unidad inferior que el tamaño del *array*.
En nuestro programa utilizamos una variable entera, *dia*, como índice para referirnos a los diferentes elementos del *array*. Esta variable



FOTOGRAFIA 1: INSTRUCCIONES DE MANEJO DE BITS.

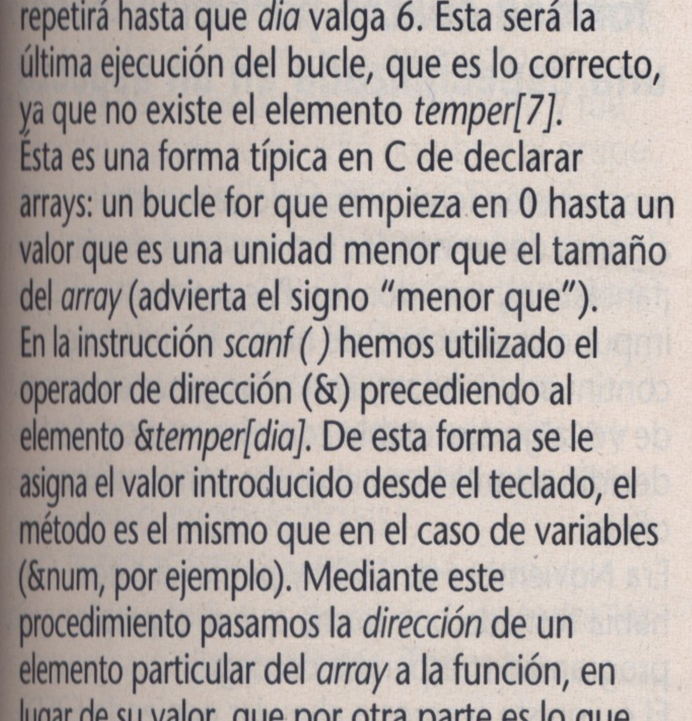
puede tomar el valor que queramos, y por lo tanto, puede apuntar por turno a los diferentes elementos del *array*. La posibilidad de utilizar variables como índices es la que hace interesante el empleo de *arrays*.

INTRODUCCIÓN DE DATOS EN UN *ARRAY*

Veamos la parte de código encargada de introducir los datos en el *array*:

```
for(dia= 0; dia<7; dia++)      /*pone
temperaturas en array */
    {
     printf("Introduzca la temperatura para el
día %d: ",dia);
     scanf("%d", &temper[dia]);
    }
```

El bucle *for* es el encargado de repetir siete veces la pregunta, almacenando, además, las temperaturas introducidas por el usuario. La primera vez que se ejecuta el bucle, *dia* tiene el valor 0, por lo que la instrucción *scanf( )* almacenará el valor tecleado en el elemento *temper[0]*, primero del *array*. Este proceso se repetirá hasta que *dia* valga 6. Ésta será la última ejecución del bucle, que es lo correcto, ya que no existe el elemento *temper[7]*.
Ésta es una forma típica en C de declarar arrays: un bucle for que empieza en 0 hasta un valor que es una unidad menor que el tamaño del *array* (advierta el signo "menor que").
En la instrucción *scanf ( )* hemos utilizado el operador de dirección (&) precediendo al elemento *&temper[dia]*. De esta forma se le asigna el valor introducido desde el teclado, el método es el mismo que en el caso de variables (&num, por ejemplo). Mediante este procedimiento pasamos la *dirección* de un elemento particular del *array* a la función, en lugar de su valor, que por otra parte es lo que necesita *scanf( )*.



MAPA DE LAS INSTRUCCIONES DE DESPLAZAMIENTO.

LECTURA DE LOS DATOS CONTENIDOS EN EL 'ARRAY'

El programa anterior lee los datos del *array* y los utiliza para calcular la media. El segundo bucle *for*, cuya expresión es idéntica a la del primero, añade el valor de las temperaturas de cada día a una variable llamada *suma*. Una vez que se han sumado todas las temperaturas, se divide el resultado entre 7.

```
suma = 0;
  for(día=0; dia<a7; dia++)
      suma+= temper[dia];
printf("La media es %d.", suma/7);
```

EMPLEO DE OTROS TIPOS DE VARIABLE

Aunque el ejemplo anterior emplea un *array* de tipo *int*, los *arrays* pueden ser de cualquier tipo. Como ejemplo reescribiremos el programa temp.c con un *array* de tipo *float*.

```
/* fltemp.c */
/* media semanal de las temperaturas */
main( )
```

Estas 13 instrucciones se dividen en aritméticas y lógicas

```
{
 float temper[7];     /* declaración del array*/
 float suma;
 int dia;

 for(dia= 0; dia<7; dia++)      /*pone
temperaturas en array */
    {
     printf("Introduzca la temperatura para el
día %d: ",dia);
     scanf("%d", &temper[dia]);
    }
 suma = 0;
 for(día=0; dia<a7; dia++)
     suma+= temper[dia];
  printf("La media es %.1f.", suma/ 7.0);
}
```

LECTURA DE UN NÚMERO DESCONOCIDO DE ELEMENTOS

¿Qué ocurre si, en principio, no conocemos el número de elementos de información que deben ser almacenados en el *array*? Veamos un programa que acepta cualquier número de temperaturas (hasta 40) y luego calcula la media:

```
/* fltemp2.c */
/* media de un número arbitrario de
temperaturas  */
#define LIM 40
main( )
 {
  float temper[LIM];
  float suma = 0.0;
  int num, dia = 0;

  do
    {
     printf("Introduzca la temperatura para el día
%d: ", dia);
     scanf("&f", &temper[dia]);
    }
  while (temper[dia++] >0 );

  num = dia -1;
  for(dia = 0; dia<num; dia++)
      suma += temper[dia];
  printf("La media es %.1f", suma/num);

 }
```

Como puede ver, hemos reemplazado el bucle *for* por un bucle *do while*. Este bucle le pide repetidamente al usuario que introduzca una temperatura y luego almacena la respuesta en el *array* temper.
La forma de salir del bucle es introducir un 0 o una temperatura negativa (claramente este programa no vale para climas fríos). Cuando se introduzca el último elemento, la variable *dia* tendrán un valor una unidad mayor que el número de elementos introducidos. Esto ocurre porque tiene en cuenta el 0 (o el número negativo) que introduce el usuario para terminar la entrada. Así, para calcular el número de datos introducidos, valor que se almacenará en la variable *num*, se resta 1 a *dia*. Luego se utiliza a *num* como límite en el segundo bucle *for*, que suma las temperaturas, y también como divisor de la suma restante.
Hay otro cambio en el programa. Hemos usado una directiva *#define* para dar al identificador LIM el valor 40:

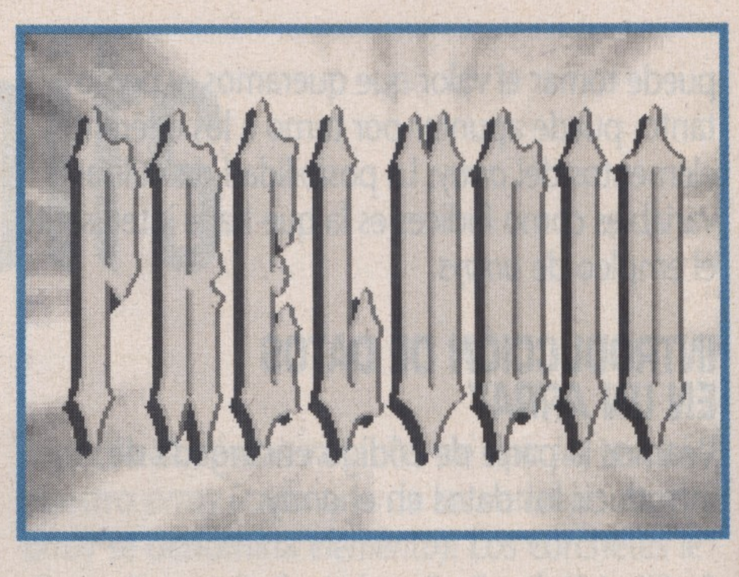
```
#define LIM 40
```

Luego hemos utilizado LIM en la declaración del *array*. El empleo de un valor con *#define*, es una práctica normal en el C. Más tarde, si queremos modificar el tamaño, todo lo que tenemos que hacer es cambiar el 40 de la instrucción ***#define***, y la modificación se hará efectiva en cualquier parte del programa donde aparezca LIM. Este programa utiliza LIM una sola vez, pero veremos pronto ejemplos en los que la dimensión del *array* aparece repetidas veces en el listado del programa, y es en estos casos donde realmente el empleo de la directiva *#define*.

JUEGOS GANADORES: 1º

Autor: José Luis Parra Lozano y otros

Preludio



Un juego de lucha en el que tendremos que escoger entre uno de los dos personajes protagonistas. Una vez hecho, tendremos que salir a los coloristas escenarios a medir nuestras fuerzas contra los variados adversarios que se nos presentarán constantemente. Un juego con sabor a recreativas. Enhorabuena a NeQ Productions.

Hay que decir que este juego no tiene completo el código fuente, la explicación la dan los propios creadores del título al final del artículo y es una excusa más que razonable. Cuando tengamos el código fuente completo lo meteremos en el CD-Rom que acompañe a la revista en números futuros para que podáis acceder al código de este magnífico trabajo.
Esta es la información que nos ha suministrado este grupo de cuatro programadores sobre su trabajo *Preludio*, os la ponemos tal cual:

PRESENTACIÓN

NeQ PRODUCTIONS es un grupo de desarrollo que abarca varios campos artísticos de producción propia. Entre ellos se encuentra el desarrollo de videojuegos. Pero también se dedica a la realización de cómics. Estos son actualmente los campos con más vistas de explotación, aunque también se está manejando la idea de la creación de CD's de Música Informatizada. Los tres campos de desarrollo estarán bajo el sello de NeQ PRODUCTIONS, así como algún otro tema que se decida realizar en el seno del grupo.
En la actualidad, el grupo se compone de 4 miembros:

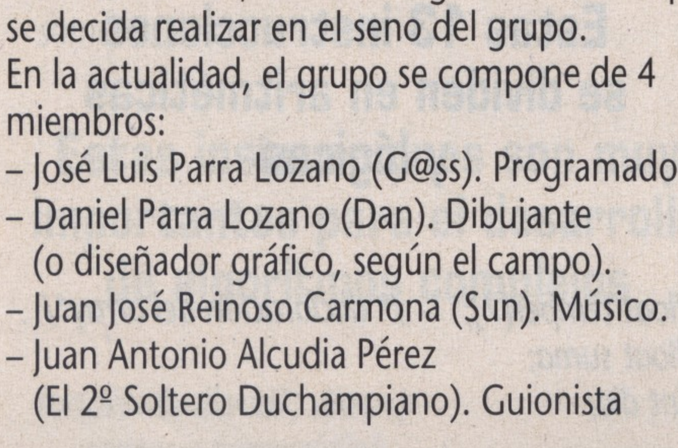
- José Luis Parra Lozano (G@ss). Programador.
- Daniel Parra Lozano (Dan). Dibujante (o diseñador gráfico, según el campo).
- Juan José Reinoso Carmona (Sun). Músico.
- Juan Antonio Alcudia Pérez (El 2º Soltero Duchampiano). Guionista

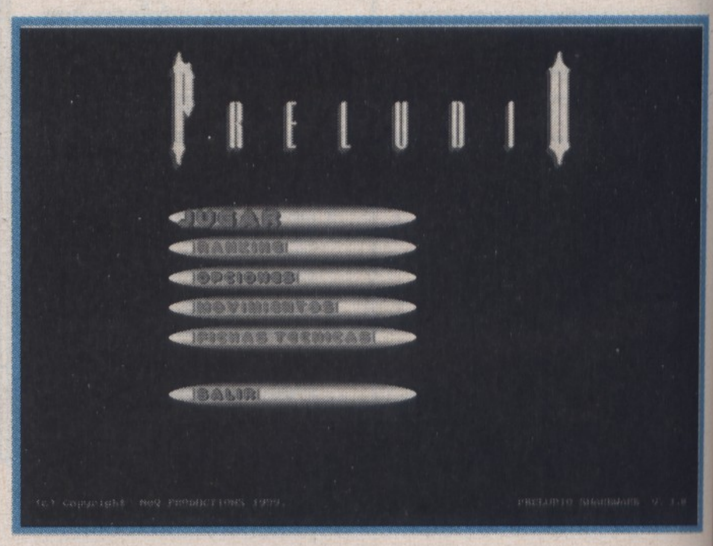
Todo comenzó con la presentación del nuevo proyecto en cómic: "PreludiO". Y fue entonces cuando se decidió hacer también el videojuego basado en el mismo. Era noviembre de 1998, proyectos que no se comenzaron apenas hasta septiembre de 1999. En este momento todavía no se había fundado grupo alguno, estaban G@ss y Dani Parra prácticamente haciendo sus trabajos por separado sin mucha motivación, pues el proyecto parecía no avanzar. Entonces apareció Sun y nos mostró unos temas musicales que había compuesto para el juego, parecía que estaba más ilusionado que los propios creadores, y cuando se escuchó algunas de las bandas sonoras que ofrecía: ¡fantástico!, son idóneas. Fue realmente el impulso que motivó de nuevo la ilusión de continuar y avanzar, con unas ganas enormes de ver algo ejecutable con esas músicas. Se decidió admitirlo en el grupo como músico oficial.

Este grupo de programación lo forman cuatro personas, cada una especializada en un aspecto

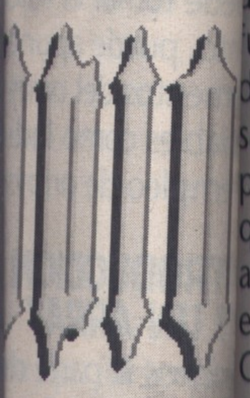
Era Noviembre de 1999 y por fin el juego había tomado una forma que ni el propio programador esperaba conseguir.
El guionista (vamos a abreviar poniendo E2SD), apareció escribiendo algunas historias (buenas historias) para realizarlas en Cómic, que se desarrollaban paralelamente con "PreludiO Comic", pero hasta el momento no había colaborado en el proyecto (tanto cómic como videojuego), tan sólo se le había comentado

> 23. Las provisiones que obtuvimos con el intercambio comienzan a escasear antes de lo previsto. Espero que no estemos muy lejos de la próxima ciudad.
>
> (...) Esta mañana he llorado. Proseguíamos nuestro camino hacia la ciudad, cuando divisamos una silueta quebrada en la distancia. Al acercarnos, hemos descubierto los escombros de una antigua casa devorados por la arena y los lagartos. En medio de ellos, se retorcía un árbol petrificado. Su tronco estaba torcido y sus ramas afiladas se eri-





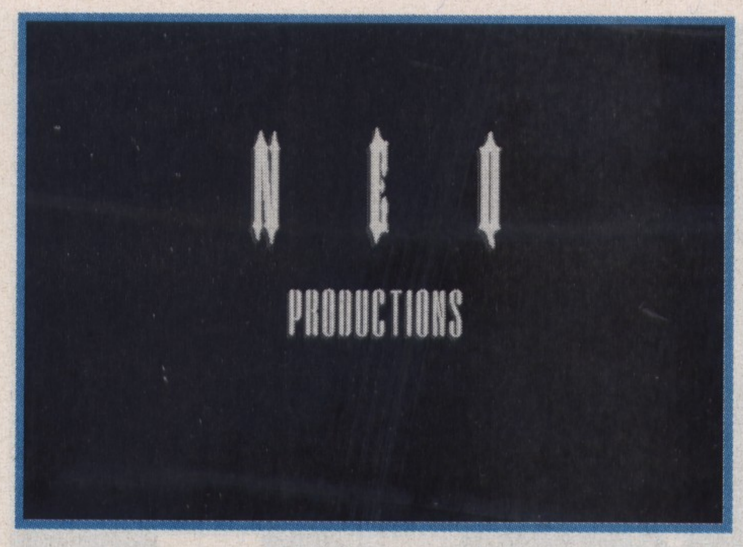
ozano y otr



que se estaba llevando a cabo, hasta que vio una prueba preliminar del juego: ¡esto tiene futuro! Y cada vez que visitaba el lugar de desarrollo observaba los progresos: ¡esto se sale! La historia de "PreludiO" estaba creada, pero hacía falta una introducción en el cómic, que resultara impactante, que fuese diferente al simple relato de la historia, tarea que fue encargada al guionista particular de Dani. Cuando mostró su obra: ¡fascinante!, ¡qué calidad, qué juego de palabras!, ¡buenísima! Entonces se pensó en utilizarlo también como intro para el videojuego, y así se hizo, y los resultados no pudieron ser mejores. En ese momento se consagró como guionista oficial del grupo. Ya estaba todo, al menos lo básico. Había programador, grafista, músico y guionista. Lo único que hacía falta era un nombre para el grupo.

Pues bien, antes de "PreludiO", Dan había realizado otro cómic compuesto de dos números, un cómic que lo ha dado a conocer a mucha gente del entorno, que incluso ha sido comentado en la revista KAME nº 32, su nombre es "NeQ". Así, después de 2 años de haber sacado el número 1, todavía hay quien pide copias. Viendo el éxito del cómic y que está siendo tan conocido, se pensó que había que explotar al máximo ese momento, no todas las obras son aceptadas tan bien, y fue entonces cuando se decidió nombrar al grupo con el nombre de "NeQ PRODUCTIONS", nombre que fue muy bien acogido por todos aquellos adictos de "NeQ". Nacía a principios de Noviembre de 1999. Por lo tanto, los Estudios Gass, que así se etiquetaban los videojuegos, cuando su componente único y principal era G@ss, deja su nombre para dar paso a "NeQ PRODUCTIONS".

NeQ PRODUCTIONS es un grupo serio, que desarrollará productos de calidad, en todos los campos y aspectos cubiertos. Todos sus



productos estarán cuidados al máximo detalle posible para que sean disfrutados por todo aquél que lo desee.

Preludio es un juego ambientado en un futuro hipercontaminado y donde la gente vive hacinada

INSTALACIÓN DEL JUEGO

Para instalar el juego hay que seguir los siguientes pasos:

1.- Insertar el CD en el lector de CD-ROM.
2.- Ejecutar el archivo INSTALL.EXE.
3.- Seguir los pasos del proceso de instalación.

Por defecto, el juego se instalará en el directorio:
C:\NEQGAMES\PRELUDIO
Si no es el directorio deseado, escribe otra ruta en el lugar correspondiente.
Terminada la instalación, se ejecutará automáticamente el setup de sonido. Si deseas volver a configurar el sonido más adelante, ejecuta el archivo "SETUP.EXE", que se encontrará en el directorio donde se ha instalado el juego.
Por fin, para jugar, ejecuta el fichero "PRELUDIO.EXE".

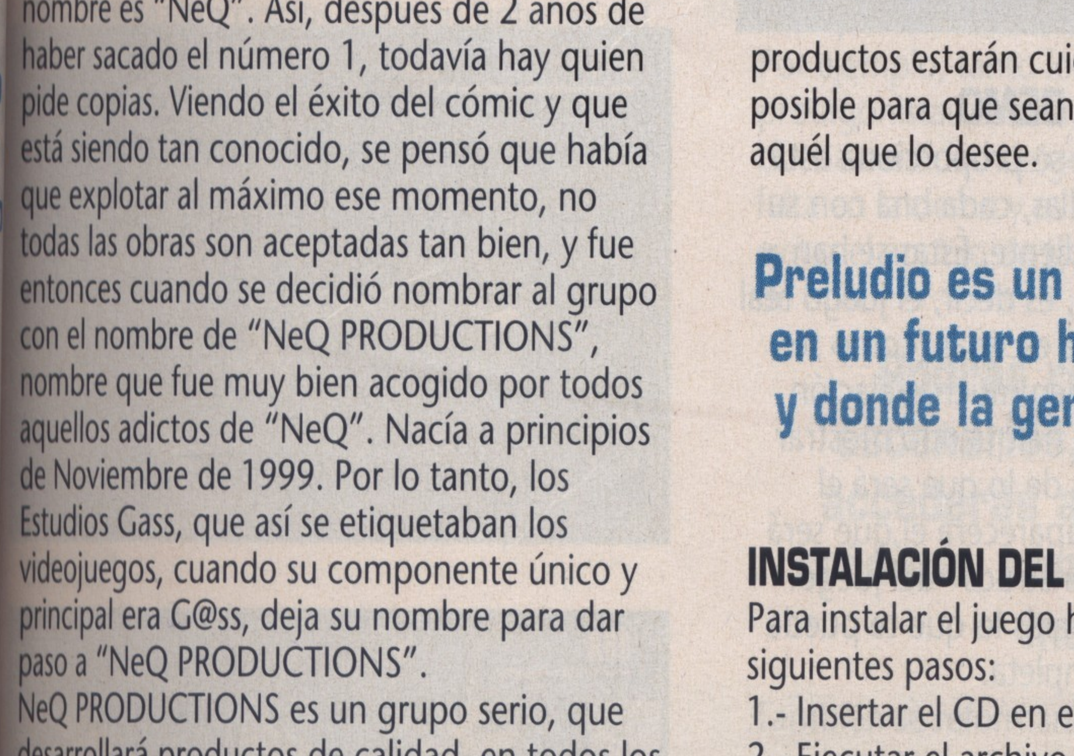
REQUISITOS

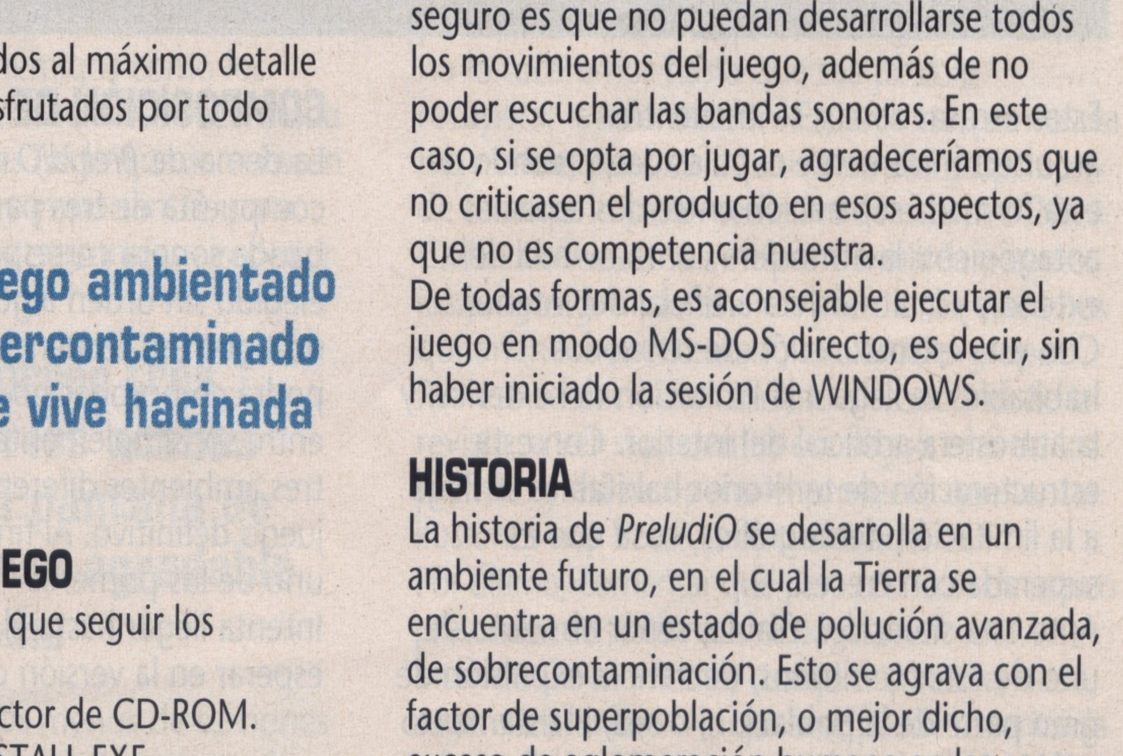
Sistema Operativo: MS-DOS.
Capacidad libre en disco: 20 Mb.

Bueno, la verdad es que los requisitos son algo que aún no han sido comprobados con exactitud. Para dar una idea, el juego se ha desarrollado en un Pentium 166 Mhz con 64 Mb de Ram.
En un ordenador lento y con poca Ram, lo más seguro es que no puedan desarrollarse todos los movimientos del juego, además de no poder escuchar las bandas sonoras. En este caso, si se opta por jugar, agradeceríamos que no criticasen el producto en esos aspectos, ya que no es competencia nuestra.
De todas formas, es aconsejable ejecutar el juego en modo MS-DOS directo, es decir, sin haber iniciado la sesión de WINDOWS.

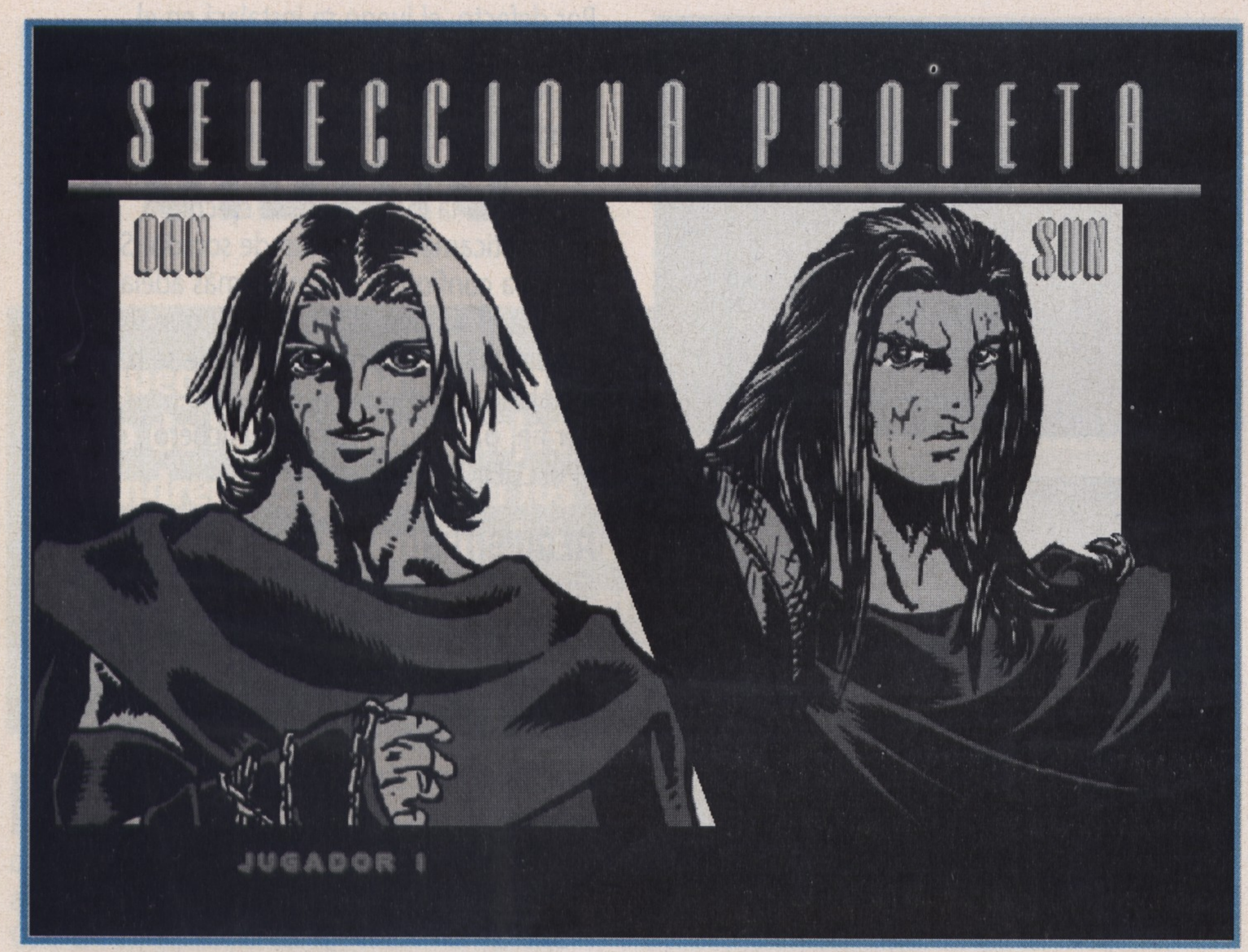
HISTORIA

La historia de *PreludiO* se desarrolla en un ambiente futuro, en el cual la Tierra se encuentra en un estado de polución avanzada, de sobrecontaminación. Esto se agrava con el factor de superpoblación, o mejor dicho, exceso de aglomeración humana en las escasas zonas territoriales que quedan habitables.





así es como él llama a los habitantes de las minas. He conseguido colarme en las minas aprovechando un descuido de vigilancia. En aquel lugar he conocido el espanto. La gente se muere de hambre y es maltratada salvajemente. Los obligan a trabajar hasta la muerte. En este infierno subterráneo, muy pocos conservan la cordura. Algunos corren semidesnudos por las galerías con la cara ennegrecida y el pelo revuelto como salvajes. Uno de estos locos se acercó a mí. Sudaba. Su rostro estaba transfigurado por la fiebre. Me llevó hasta un habitáculo que



Estas escasas zonas, se encuentran arquitectónicamente cupulizadas, creando de esta forma la separación entre dos espacios antagónicos: la atmósfera contaminada del exterior, y la atmósfera artificial del interior.
Con esta estructuración de territorios habitables se llega a la limitación del exterior, y la atmósfera artificial del interior. Con esta estructuración de territorios habitables se llega a la limitación demográfica, cosa que es superada con creces.
Ante esta situación, Cíablo, señor absoluto de uno de estos territorios, decreta la expulsión de gran parte de la población, o sea, el estamento más pobre, el cual se convertirá en la llamada raza árida, en contraste con la raza sometida a las condiciones infrahumanas del exterior, ganándose la supervivencia a cambio de trabajar duramente, bajo controles militares de Cíablo, en la extracción de materiales primarios. Esta situación se hace extrema cuando Cíablo planifica el proyecto D.A., que consiste en la eliminación de los bebés de la raza exterior, consiguiendo de forma indirecta la eliminación de la raza árida, que de lo contrario se convertiría en amenaza.
Cuentan algunas profecías que, cuando el hombre llegase a los límites de la pérdida de los derechos humanos, aparecerían dos profetas, el "preludio" de un cambio situacional.
Existe un hombre que ha llegado a esos límites: Cíablo.
Existen dos profetas: Dan y Sun.
Da comienzo la historia.

COMPOSICIÓN DE LA DEMO

La demo de *PreludiO* que se proporciona está compuesta de tres pantallas, cada una con su banda sonora correspondiente. Éstas se han elegido sin orden alguno, es decir, el juego real no sigue esa secuencia de escenas (como se podrá comprobar no tienen mucha relación entre sí), simplemente se ha querido mostrar tres ambientes diferentes de lo que será el juego definitivo. Al final aparecerá el que será uno de los primeros "monstruos" del juego. Intenta llegar hasta él y verás lo que te puede esperar en la versión completa.

La música, que destaca en este juego, es un aspecto que no se cuida demasiado algunas veces

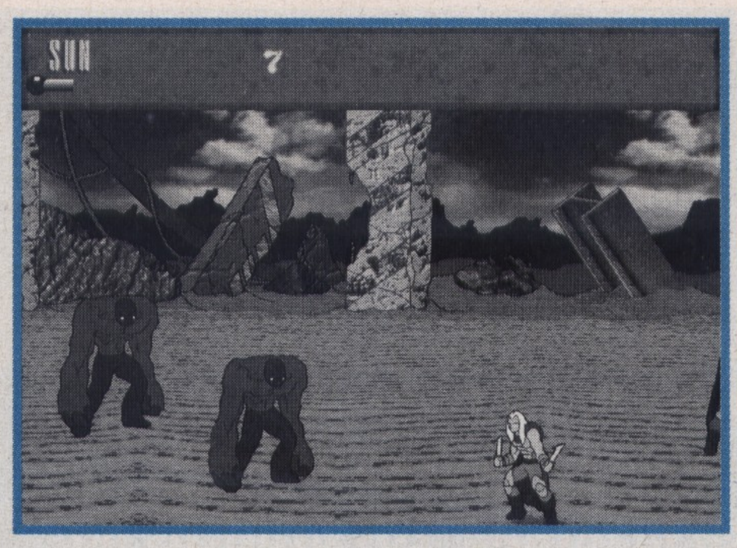
Si quieres ver los créditos, pásate la Demo completamente y los obtendrás (contiene una banda sonora diferente).
El menú de opciones también está limitado, permitiendo únicamente jugar, salir, y modificar opciones como el volumen de sonido, número de jugadores, o la configuración del teclado.

CONTROL

El funcionamiento del juego no es nada complicado, sobre todo si eres de los que se han dejado "las pelas" en las recreativas con juegos tales como *Golden Axe* entre otros. Se maneja con el teclado con sólo 6 teclas:

- Desplazamientos cursores.
- Salto.
- Disparo.

El menú principal funciona con los cursores (flechas) arriba y abajo para cambiar de opción y Enter para aceptar la opción seleccionada.
En el menú "opciones", se utiliza de forma similar los controles anteriores, salvo que en la opción para elegir el número de jugadores, se puede cambiar el mismo, tanto si se pulsa "Enter" como los cursores a derecha o izquierda. Estos cursores son los que se utilizan también para aumentar o disminuir el volumen del sonido, si esta opción es la que se encuentra seleccionada.
Al configurar el teclado aparecerá la configuración de un solo jugador o de dos jugadores, según la opción que esté seleccionada. La configuración de estos dos modos de juego son independientes, es decir, en el modo de dos jugadores, la configuración



del jugador 1 puede ser diferente, si se desea, a la configuración que tenga el modo de un jugador.
Para cambiar los controles sólo hay que pulsar la tecla que se quiera introducir para ese control, cuando su casilla se encuentre iluminada. Si no se desea cambiar una tecla, se debe pulsar la misma tecla que aparezca en esa casilla para saltarla (una pega de la configuración que se va a mejorar en la versión definitiva).
Para desplazarte por los distintos botones del menú de configuración del teclado pulsa la tecla TABULADOR, que alternativamente irá seleccionando los botones. Para aceptar un botón, pulsar "Enter".
La opción de teclado por defecto se ha tomado como óptima para evitar el malestar de jugar 2 personas en el mismo teclado, y se puede tomar cuando se desee.
Nota: Para que la configuración del teclado tenga efecto, habrá que guardar previamente los cambios con el botón "Teclado por defecto".
La pantalla de selección de personajes se maneja con el teclado del jugador1. Pulsando los cursores a derecha e izquierda, se podrá cambiar de jugador. Para seleccionar un personaje, cada jugador pulsará el botón "A" o "B", y así seleccionará el personaje cuyo título en la parte inferior de la pantalla coincida con el jugador 1 ó 2, según corresponda.

COMIENZA EL JUEGO. CONTROLES

Cursores para desplazarse por la pantalla.
- Botón A: salto.
- Botón B: golpe.
- Botón A + B: ataque especial.
- A+ (en el aire) B: patada.
- A+ desplazamiento + (en altura máxima) B: patada en picado.
- 2 veces cursor derecha/izquierda: correr a derecha/izquierda.
- Correr + B: ataque Kame.
- Correr + A + (en el aire) B: super ataque.

¿Mucho?, en realidad lo único que debes recordar es que puedes saltar, golpear y correr, los ataques los averiguarás mientras juegas.
Al pasar de pantalla, la energía de los profetas aumentará en un 30% aproximadamente, aprovéchala.
Para abandonar el juego en mitad de la partida (espero que no lo hagas, y disfrutes), pulsa Escape, y después confirma la huída.

MEJORAS FUTURAS

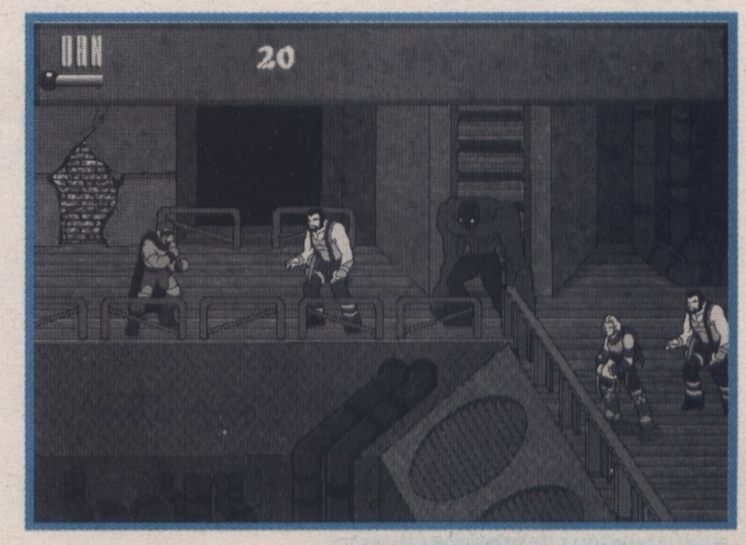
Aún así, el juego sufrirá intensas mejoras en su versión final, pues hay algunos aspectos que han quedado en el aire.
Por ejemplo, tanto los enemigos como los protagonistas se caerán al suelo tras recibir una buena cantidad de golpes (algo que se echa en falta en la demo y es muy importante), para evitar el agobio de tantos personajes alrededor de uno.

Podrás recorrer tres escenarios, si los acabas accederás a la pantalla de créditos, con otra agradable melodía

También se reformarán los menús de opciones y se intentará optimizar al máximo el uso de recursos, tanto en el aspecto gráfico como en el sonoro. Se incluirá además interactividad con el ratón.
Comentamos los cambios porque sabemos las críticas que se pueden hacer al juego, y así avisar de antemano que los fallos o defectos de esta demo, serán totalmente controlados en la versión definitiva.

COMENTARIO FINAL

Por último, queremos hacer mención de un gran problema técnico surgido justo antes de tener la demo ejecutable. Por razones desconocidas, un sábado por la mañana en Noviembre de 1999, el disco duro, en el que se encontraba todo el material de meses y meses de trabajo, fue atacado inesperadamente en la FAT, con lo que se ha quedado inutilizado. La última copia de seguridad del juego se realizó una semana antes de la tragedia. Pero en ese tiempo, el desarrollo del mismo avanzó en más de un 60%, lo cual quiere decir que dicha copia sirve más bien de poco, casi para volver a empezar. Ni siquiera el código tenemos, ni los gráficos, nada, tan sólo esta demo.
¿Se puede arreglar y recuperar la información? Sí, se puede, pero los fondos del grupo aún están en números rojos y nos es imposible costearlo, con lo que el juego lleva un paro de 2 meses.
También hay programas que recuperan la información que, a pesar de no ser excesivamente caros, escapan a nuestras posibilidades. Aún así, el disco sigue intacto, y no lo tocaremos hasta que sea posible recuperar la información. Esperamos que no sea muy tarde, pues las intenciones del grupo son de terminarlo para mediados de este año 2000.
Cada dos meses se compra Divmanía y vemos los juegos ganadores. "un premio de estos nos quitarían los problemas", exclama el grupo.
Por eso se decidió mandarlo al concurso de la revista, a ver si podemos conseguir uno de los premios y así recuperar el proyecto que tantas ilusiones nos había proporcionado.
Por último, comentar que para primeros de marzo pretendemos abrir nuestra página web en Internet. De momento, cualquier consulta, duda, sugerencia, crítica o lo que sea, puede ser enviada a la siguiente dirección mediante correo electrónico, y así os podremos mantener informados sobre el juego, así como de otros proyectos. Sería agradable conocer la opinión de los usuarios sobre esta Demo, así sabríamos mejor cómo contentarlos para las próximas versiones. La dirección es la siguiente: i62paloj@uco.es
Nada más, esperamos que disfrutéis del juego.
Saludos de NeQ PRODUCTIONS. 2000 ©



Autor: Germán Bermejo Yunq

99-00

Por fin, una aventura gráfica se ha presentado al concurso de DIVmanía, y se ha llevado el segundo premio, esperemos que no sea la última. Los gráficos no son lo que se dice una obra de arte multimedia, pero son un correcto marco para que nos metamos en el pellejo del protagonista de este juego. Enhorabuena Germán y sigue así.

Germán nos confiesa que el único tipo de juegos que le gustan son las aventuras conversacionales y por eso se ha animado a programar un título de este género. Parece que este programador autodidacta se ha "currado" este juego él solito, un gran mérito teniendo en cuenta la dificultad de elaborar un buen guión, hacer los gráficos y, sobre todo, programar un juego de estas características.

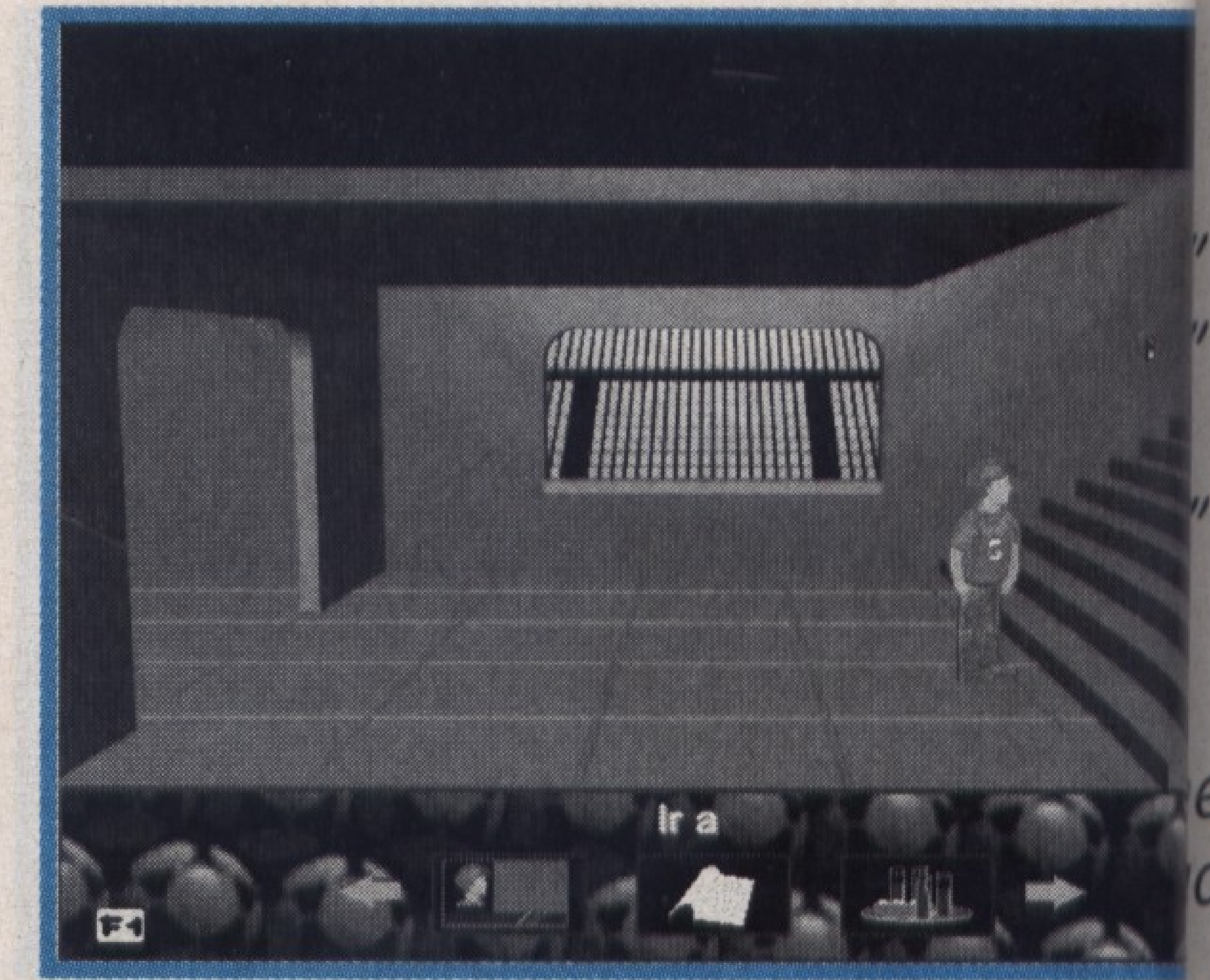
Lo que nos ha mandado es la segunda parte del juego, aunque ya tiene prácticamente terminado el guión completo del mismo. Germán lanza un grito al aire busca de gente que le ayude a llevar a buen término su proyecto, ya que no dispone de todo el tiempo que quisiera.

En esta segunda parte manejamos un personaje que trabaja en un campo de fútbol y debe resolver una compleja trama para encontrar a los jugadores perdidos. Dentro del estadio hay muchos personajes, algunos bastante conocidos, que le irán dando pistas para resolver el acertijo.

El juego se maneja con el ratón, dejando los cursores para mover el menú de objetos. Con F1 se accede a un menú, que no está terminado, en el que se puede leer un resumen de la trama del juego.

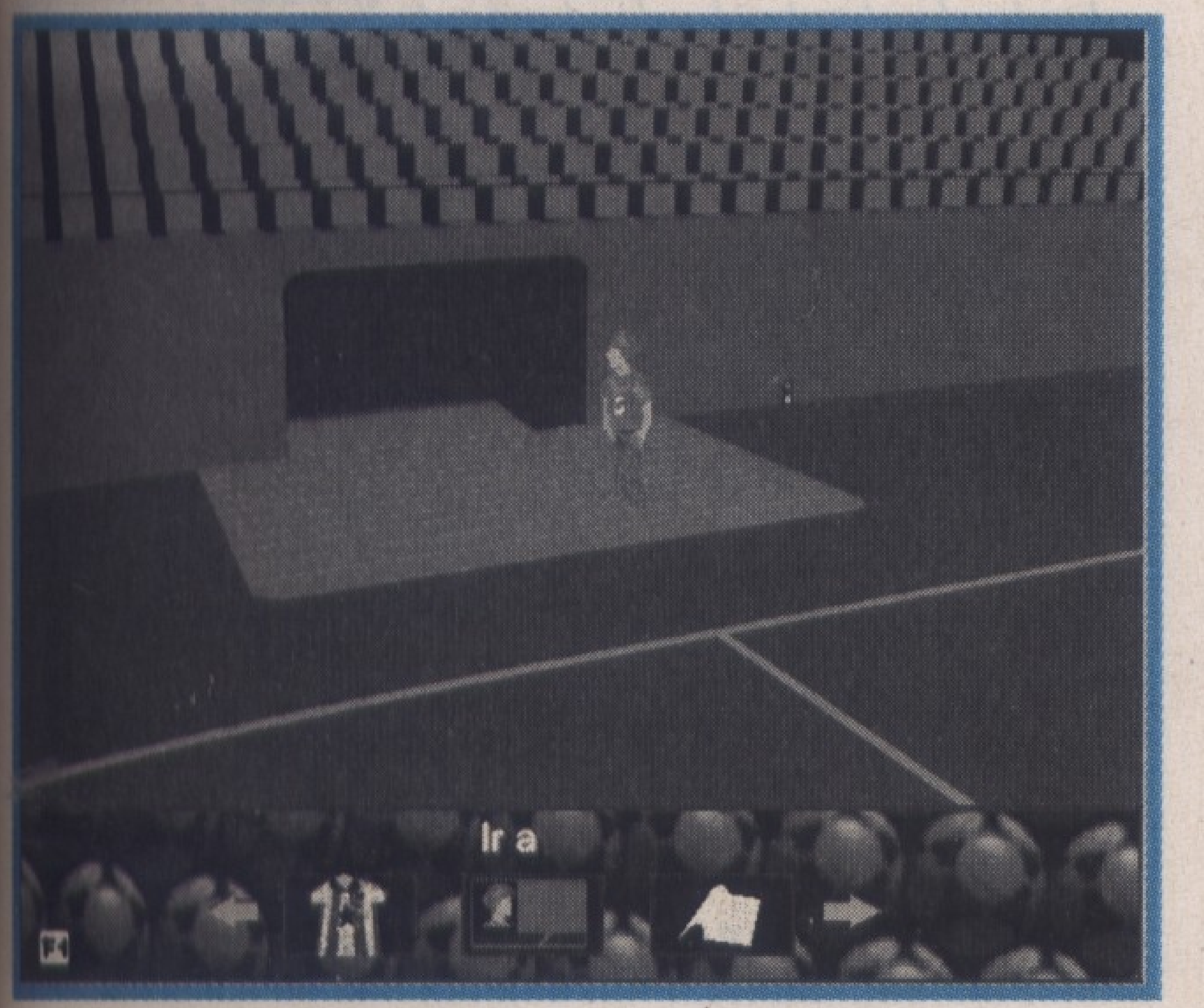
CODIGO FUENTE

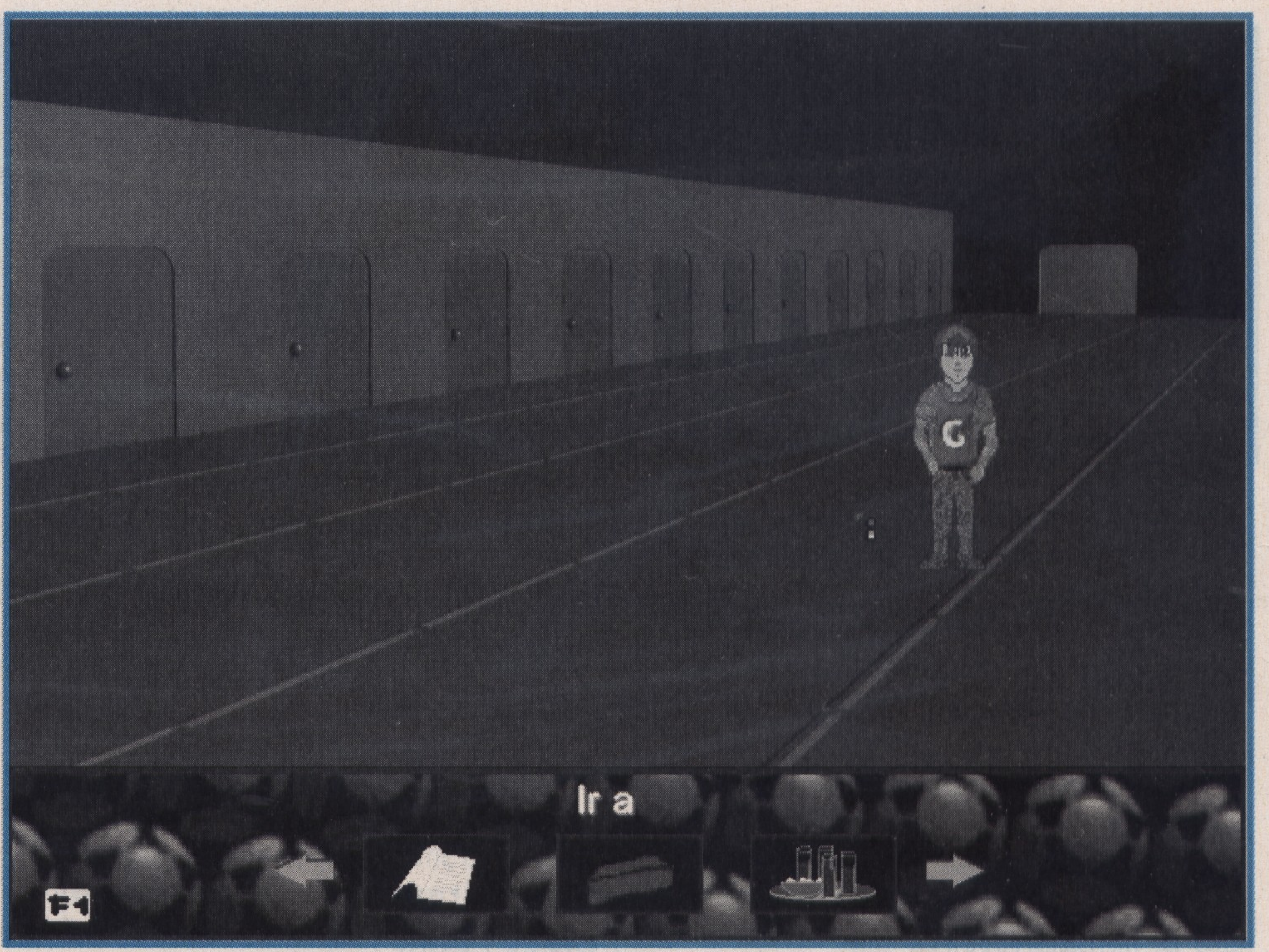
```
PROGRAM x9900;

GLOBAL
  letras[]=
        "a","b","c","d","e",
        "f","g","h","i","j",
        "k","l","m","n","¤",
        "o","p","q","r","s",
        "t","u","v","w","x",
        "y","z",".","-"," ",
        "0","1","2","3","4",
        "5","6","7","8","9",

        "A","B","C","D","E",
        "F","G","H","I","J",
        "K","L","M","N","¥",
        "O","P","Q","R","S",
        "T","U","V","W","X",
        "Y","Z",":","_",",",
        "!","","'","?","$",
        "%","&","/","(",")"," ";

  STRUCT dialogos[99];
     letras[99];
  END =10000 dup (" ");

  STRUCT nombre[714];
     letras[74];
  END =53625 dup (" ");
```




```
  idwriteletras[250]; // Se usa para recoge
del texto de cada letra

fich_dialogos[]=          "",       "",
"","room03.dia","room04.dia",
                  "",       "",       "",
"",      "",

"","room11.dia","room12.dia",          "",
"",
           "room15.dia",        "",
"",      "",

"","","","","","","","","","","","","","",
"","","","",

"","","","","room44.dia","","","","","";

fich_ordendia[]=          "",       "",
"","room03.org","room04.org",
                  "",       "",       "",
"",      "",

"","room11.org","room12.org",          "",
"",
```

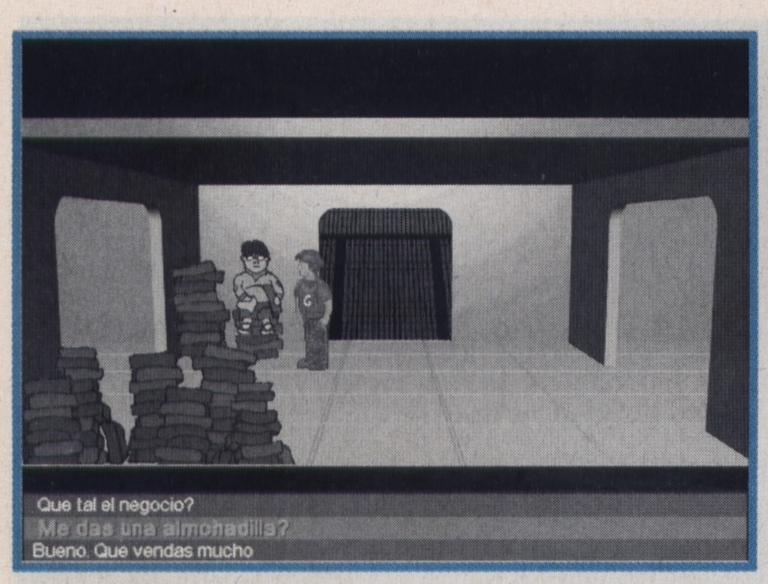
```
        "room15.org",       "",        "",
"",     "",

"","","","","","","","","","","","","","","","",
"","","","",

"","","","","room44.org","","","","","";

fichero[44]=   "calle.pcx",    "cuarto.pcx",
"cuarto2.PCX", "habicoke.pcx",
          "habi6.pcx",     "habi6.pcx",
"esca2.pcx",   "esca3.pcx",
        "cabinas1.pcx",  "cabinas2.pcx",
"antesala.pcx", "vestuari.pcx",
          "ante2.pcx",     "cesped.pcx",
"palco.pcx",  "tienda2.pcx",
         "grada1.pcx",    "grada2.pcx",
"grada3.pcx",   "grada4.pcx",
         "grada5.pcx",    "grada6.pcx",
"palquito.pcx", "palquito.pcx",
        "palquito.pcx",  "palquito.pcx",
"palquito.pcx", "palquito.pcx",
        "palquito.pcx",  "palquito.pcx",
"palquito.pcx", "palquit2.pcx",
          "cabina.pcx",    "cabina.pcx",
"cabina.pcx",   "cabina.pcx",
          "cabina.pcx",    "cabina.pcx",
"cabina.pcx",   "cabina.pcx",
          "cabina.pcx",    "cabina.pcx",
"cabina.pcx",   "cabina.pcx",
         "goligol.pcx";

factor[]=
133,100,100,100,100,100,100,100,609,609,
     177,385,301,135,  0,135, 15, 15, 15,
15,
     15,  0,  0,  0,  0,  0,  0,  0,  0,  0,
      0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
      0,  0,  0,  0,  0,  0,  0,  0,  0,  0;

tama[]=   90,90,90,90,90,90,90,90,25,25,
     84,68,84,70,70,70,26,26,26,26,
     26,26,90,90,90,90,90,90,90,90,
     90,90,90,90,90,90,90,90,99,90,
     90,90,90,90,90,90,90,90,90,90;

linea[]=
```

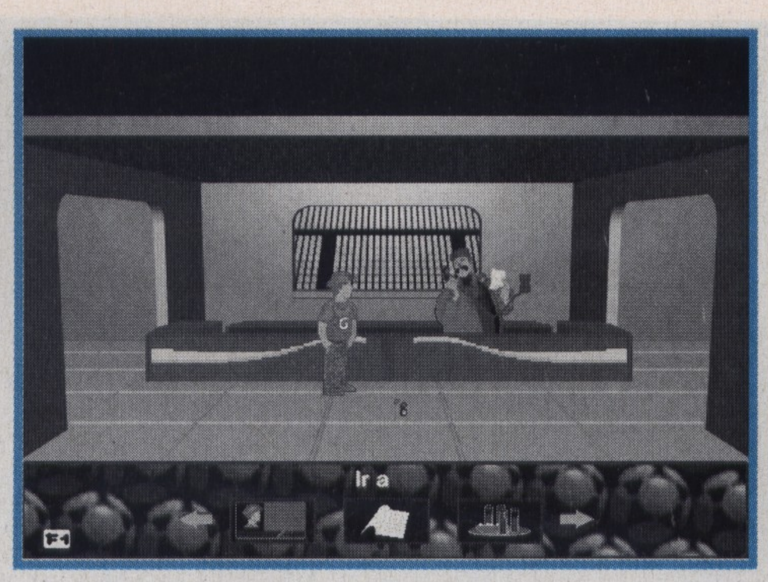


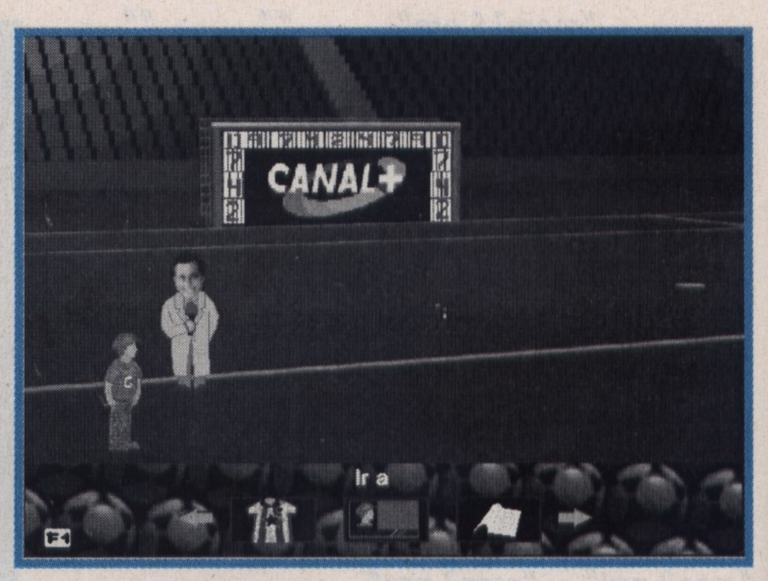



```
200,283,283,283,283,283,283,283,154,154,
        200,195,230,210,235,235, 77, 77, 77,
77,

77,235,235,235,235,235,235,235,235,235,

235,235,235,235,235,235,235,235,235,235,

235,235,235,235,235,235,235,235,235,235;

mirar_x[]=  0,  0,  0,395,207,  0,  0,  0,  0,  0,
         0,120,480,  0,  0,241,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  0,147,  0,  0,  0,  0,  0;

personaje_x[]=0,0,0,405,182,0,0,0,0,0,120,0,
0,0,0,0,0,0,0,0,0,  0,0,
          0,0,0,  0,  0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,146,0,0,0;

personaje_y[]=0,0,0,232,237,0,0,0,0,0,153,0,
0,0,0,0,0,0,0,0,0,  0,0,
          0,0,0,  0,  0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,262,0,0,0;

puerta_x[]=169,181,203,229,252,283,323,377
,460,572;
puerta_y[]=157,159,163,167,172,177,184,194
,210,230;

puerta2_x[]=481,468,452,434,413,383,353,31
1,249,166,54;

puerta2_y[]=155,157,160,163,167,173,179,18
6,197,209,230;

cog[150]=0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0,
0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0,  //29
        0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0,
0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0,  //59
        0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0,
0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0,  //89
        0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0,
0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0,  //119
        0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0,
0,0,0,0,0,  0,0,0,0,0,  0,0,0,0,0; //149

x_y_dialogos_personajes[]=0,0,     0,0,   0, 0,
280,150,  40,90,
                    0,0,    0,0,   0, 0,     0,0,
0,0,
                    0,0,  50,100, 320,50,     0,0,
0,0,
                 150, 70,    0,0,   0, 0,     0,0,
0,0,
                    0,0,    0,0,   0, 0,     0,0,
0,0,
                    0,0,    0,0,   0, 0,     0,0,
0,0,
                    0,0,    0,0,   0, 0,     0,0,
0,0,
                    0,0,    0,0,   0, 0,     0,0,
0,0,
                    0,0,    0,0,   0, 0,
0,0,100,200;

acciones_completadas[]=0,0,0,0,0,
```

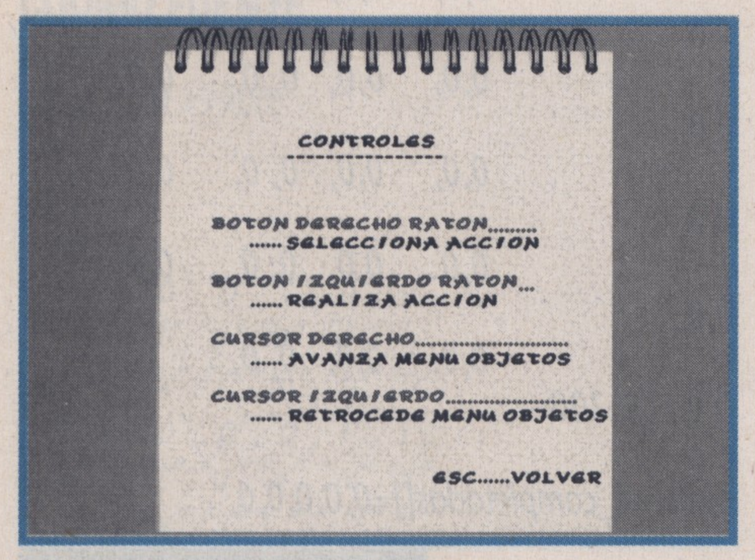
Juegos ganadores: 2º








```
//TODAS LAS ACCIONES
                0,0,0,0,0,      //QUE DEBEMOS
REALIZAR
                0,0,0,0,0,      //PARA AVANZAR
EN EL JUEGO
                0,0,0,0,0,      //0 NO
REALIZADAS
                0,0,0,0,0;      //1 SI
REALIZADAS


tabla_de_mensajes[1000];


acciones[]="Ir a ","Usar ","Mirar ","Hablar
","Coger ";

nombre_objetos[]="vacio","camiseta","bandeja"
,"tarjeta","chupa",  //NOMBRE DE LOS

"nintendo","botas","balon","papel","medicame
nto",   //OBJETOS QUE

"dinero","refrescos","sobre","gorra","globo",
//APARECEN EN

"periodico","almohadilla","cesped","lista","ban
deja sucia",//EL JUEGO
            "bola periodico mojada","bandeja
sucia","dinero y descuento",
            "botas con bola","botas
ensanchadas","botas calzadas";

ICONOS[19]=41,43,58,40,40,40,40,40,40,40,4
0,40,40,40,40,40,40,40,40,40;
            // CODIGOS DE LOS OBJETOS QUE
```

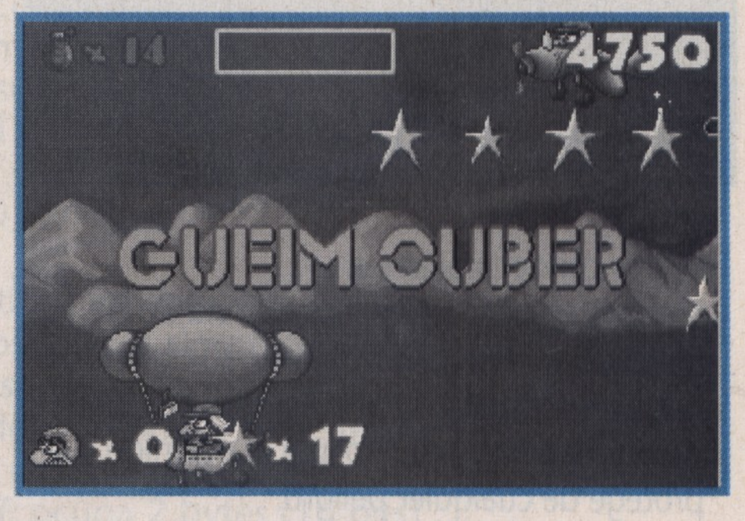



```
LLEVAMOS EN EL INVENTARIO

STRUCT habitaciones[44]     //TODOS LOS
OBJETOS DE TODAS
    objeto_1[14];           //LAS HABITACIONES

END=
"","","","","","","1","2","3","4","5","cartel","tor
nos","papeles","cartel","taquilla",

"","","","","","","1","2","3","4","5","6","7","8"
,"escalera","puerta",

"","","","","","","1","2","3","4","5","6","7","8"
,"escalera","ventana",

"jefe","","","","","bandeja","","3","4","5","6
","7","telefono","mostrador","ventana",

"Al","","","","","","1","2","3","4","5","6","7","
","almohadillas","puerta",

"","","","","","","1","2","3","4","5","6","7","8"
,"9","puerta",

"","","","","","","1","2","3","4","5","6","7","8"
,"escalera","ventana",

"","","","","","","1","2","3","4","5","6","7","8"
,"escalera","ventana",

"","","","","puerta","puerta","puerta","puerta","
puerta","puerta","puerta","puerta","puerta","puerta","p
uerta","puerta","escalera",

"","","","puerta","puerta","puerta","puerta","
puerta","puerta","puerta","puerta","puerta","puerta","p
uerta","puerta","escalera",

"","","","","","","1","2","3","4","5","6","7","8"
,"puerta","escalera",
    "eterno suplente","periodico
mojado","","","","","","3","4","5","alineacion
","pizarra","bancos","jacuzzi","puerta",

"","","","","","","","","","","","","","escalera","c
esped","ascensor",

"","","","","","","1","2","3","4","5","6","7","gr
adas","cesped","salida",

"","","","","","","bandeja","2","3","4","5","gra
das","dinero","presidentes","trofeos","ascensor"
,

"dependienta","","","","botas","balon","2","ca
ja","cliente","5","zubi","camisetas","botas","b
alon","salida",

"","asientos","asientos","asientos","asientos","
asientos","asientos","cabina","cabina","cabina"
,"cabina","cabina","cabina","palco","palco",

"","","asientos","asientos","asientos","asientos
","asientos","asientos","palco","palco","palco",
"palco","palco","palco","palco presidencial",

"","asientos","asientos","asientos","asientos","
asientos","asientos","cabina","cabina","cabina"
,"cabina","cabina","cabina","palco","palco",

"","","","","","","asientos","asientos","asiento
s","asientos","asientos","asientos","cesped","",
"salida",

"","","","","","","asientos","asientos","asiento
s","asientos","asientos","asientos","cesped","",
"salida",

"","","","","","","asientos","asientos","asiento
s","asientos","asientos","asientos","cesped","",
"salida",

"","","","","","","","","","","gradas","butacas
","","","",

"","","","","","","","","","","gradas","butacas
","","","",

"","","","","","","","","","","gradas","butacas
","","","",

"","","","","","","","","","","gradas","butacas
","","","",

"","","","","","","","","","","gradas","butacas
```

Autor: Tomás Baenas Tormo

¡Guerra!

Un juego de plataformas se ha alzado con el tercer premio de este número de DIVmanía. Sus meritos: su perfecta jugabilidad, la variedad de personajes y objetos que podemos encontrar a lo largo del juego, su completo manual de instrucciones y sus constantes dosis de humor. Si no nos creéis, sólo tenéis que instalarlo en vuestro ordenador.



Aquí tenéis un extracto de la completa guía de este juego y un trozo del código fuente del mismo. Las instrucciones y el código lo podréis encontrar entero en el CD-Rom que acompaña a la revista.

GUIA DEL JUEGO

Ayudado o no por su amigo francotirador, Máquina, alias ¡Guerra!, deberá enfrentarse a la ardua misión de recoger todas las estrellas que el enemigo ha hurtado al coronel 'Trauma' y alcanzar el punto 'Victoria' de cada nivel. No es un trabajo muy loable, pero así es la vida del soldado profesional...

LOS ENEMIGOS

1. Soldado profesional del batallón de 'osos peligrosos'.
2. Soldado incinerador de la escuadra de 'mecheros'.
3. Soldado con bazoka, escuadra de 'supositorios'.
4. Soldado de la brigada paracaidista.
5. Kamikaze.
6. Artillero del batallón de 'burros'.
7. Camión para el transporte de tropas.
8. Motoristas de vigilancia armada.
9. Bomba de fragmentación

Nota: el soldado profesional enemigo es peligroso, dispone de todo tipo de armamento así como de un conocimiento experto de las artes marciales japonesas. Es letal con sus propias manos, está entrenado para ignorar las condiciones meteorológicas, come cosas que harían vomitar a una cabra... y no tiene miedo al dolor.

ELEMENTOS DEL ESCENARIO

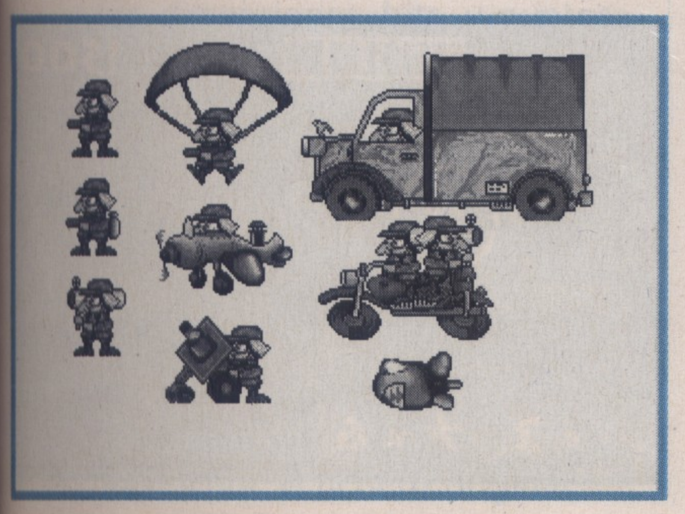
1. Barriles explosivos, rellenos de metralla.
2. Aspersores de fuego.
3. Dirigible monoplaza.
4. Teleférico.

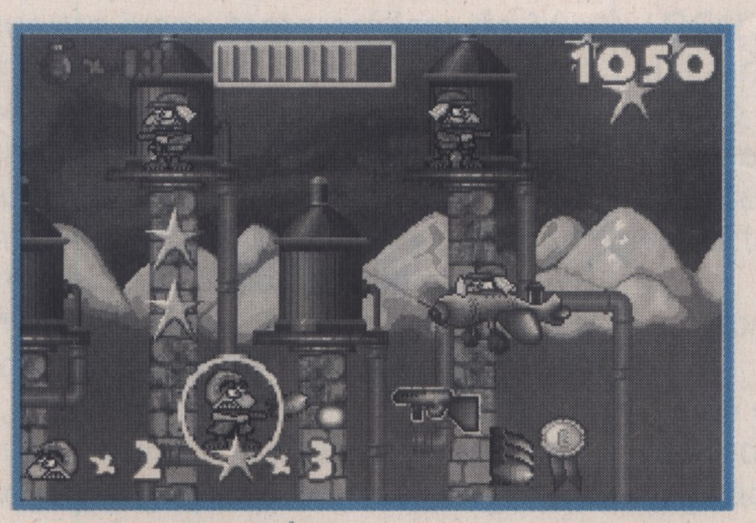
Notas: Puedes utilizar los barriles en tu propio beneficio, pero recuerda no estar muy cerca cuando procedas a su detonación (siempre y cuando no se te adelante el enemigo, claro está). Los aspersores sólo son vulnerables en la pieza agarradera. Debes utilizar estos dirigibles monoplaza de vigilancia de zonas elevadas para proseguir tu camino en aquellas regiones donde hay pequeños abismos. Todos sabemos lo que es un teleférico.

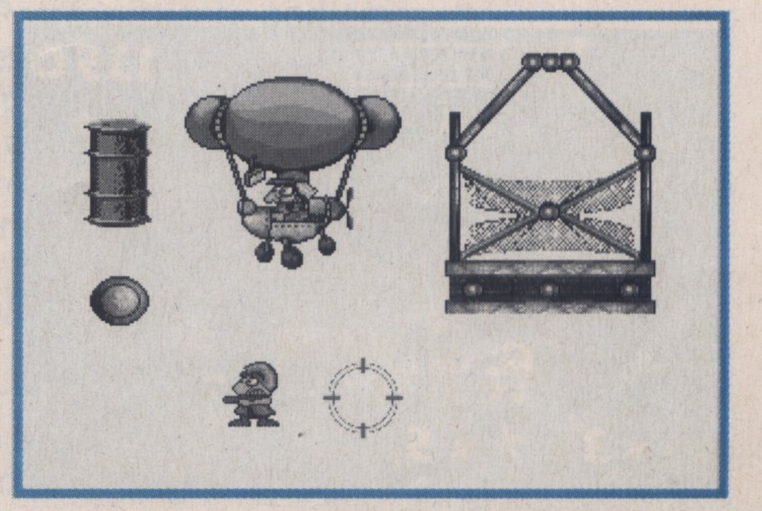
OBJETOS, ARMAMENTO Y MARCADORES

1. Subfusil ligero KK-3.
2. Estrellas del coronel 'Trauma' y marcador de las conseguidas.
3. Punto 'Victoria'.
4. Caja de granadas.
5. Ítem de vida extra.
6. Cañón pesado modelo 'cacharro'.
7. Caja de cohetes auto-guiados.
8. Condecoraciones y marcadores de habilidades especiales.
9. Botiquín de primeros auxilios.
10. Caja de munición de ráfaga.
11. Garrafa de gasolina y marcador de función incineradora.
12. Marcador de función lanzamisiles.
13. Marcador de granadas de mano.
14. Marcador de ráfagas.
15. Barra de vida de '¡Guerra!'.

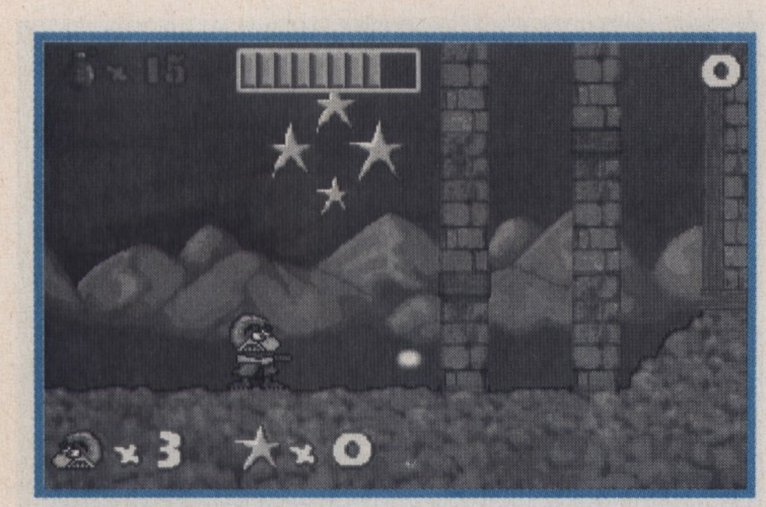
Notas: el subfusil ligero KK-3 tiene tres funciones: proyectiles convencionales (hasta triple ráfaga), incinerador y lanzamisiles autoguiados, según la recarga que se le suministre. ¡Guerra! ha de recoger todas las estrellas de 'Trauma'. Sabrá que ha completado este objetivo cuando suene una melodía a tal efecto dispuesta y observe cierta evolución en el punto 'Victoria'.
Recarga de granadas de mano temporizadas. Estas granadas son el alma de ¡Guerra!, y su utilidad trasciende más allá de la meramente destructiva. Aprende a medir su distancia de explosión y trayectoria y échale un vistazo a la sección de trucos.
Podrás hacer uso del 'cacharro' si logras arrebatárselo a su artillero propietario.

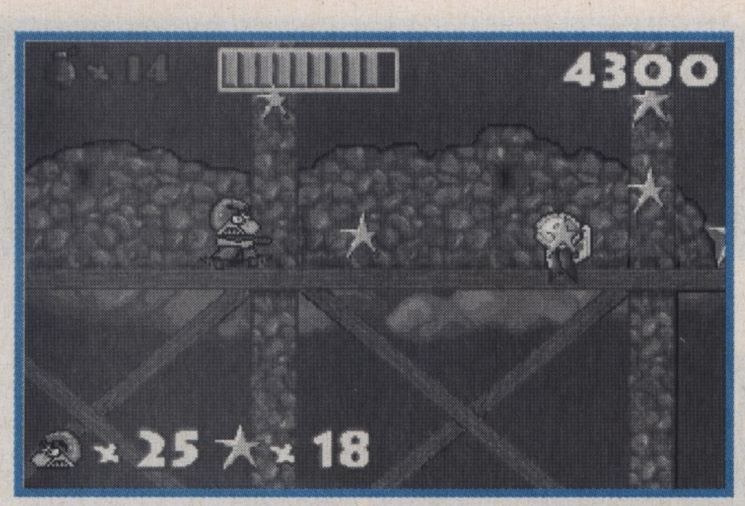






Juegos ganadores 3º





Hay tres condecoraciones:
- Medalla escudo: realmente se trata de la 'medalla al valor' del congreso, su obtención inspira en '¡Guerra !' tal complejo de superioridad que electrocuta al enemigo y le protege de cualquier peligro.
- Medalla salto: obtenida en la 'Olimpiada militar' por la habilidad de salto de altura.
- Medalla carrera: ídem pero por la habilidad de velocidad punta.

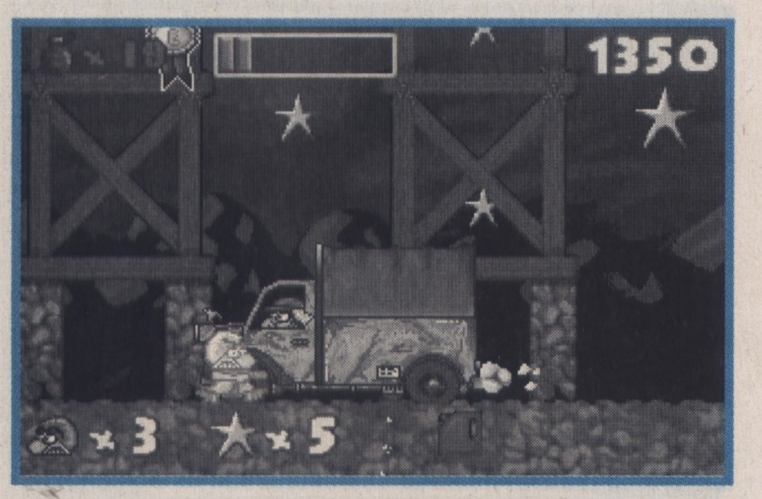
Munición para la función ráfaga del francotirador, para obtenerla tienes que dispararle.

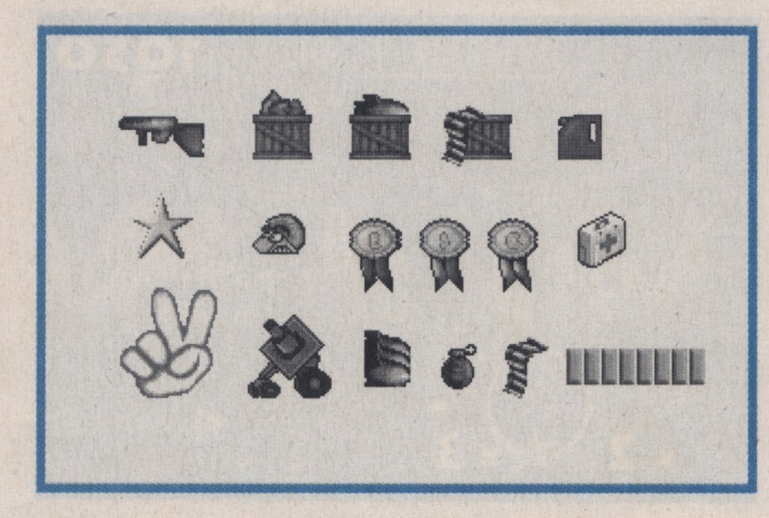
RECUERDA...

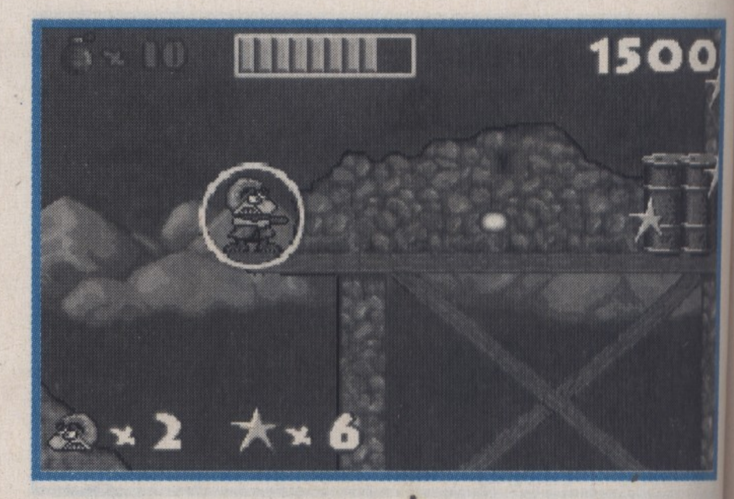
Máquina es un soldado profesional de elite, entrenado por los mejores... pero sólo es un hombre, y ésta no es su guerra. Ustedes le llamaron a él, no él a ustedes... él hará lo posible para ganar, pero puede que no le dejen ganar. Lo que ustedes llaman infierno, él lo llama hogar...
Dios no se tomó la molestia de crearlo... fui yo.

CODIGO

```
/*
  -GUERRA! (C) Tom s Baenas Tormo'2000


*/
//compiler_options _ignore_errors;
program tomi;


global

   pantalla=1;
   num_jugadores=1;
   extra=1;
   tiros=0;

   quieto=0;
   fumar=0;

   opcion=1;
   pulsa=0;
   exter;

   fin_nivel=false;
   vidas=3;
   bonos=0;
   puntos=0;
   granadas=15;

   balas=75;
   llamear=0; vel;
   dura_llamear=0;

   cacha=0;

   tipo_disp=1; //1
   cohete=0;
   dura_cohete=0;
   parte_pan=0;

   Energia=8;
   momento=0;

   pon_escudo=0;
   super_salto=0;
   dura_salto=0;
   dura_escudo=0;
   carrera=0;
   dura_carrera=0;
   vulnerable=0;
   botado=0;
   num_soldados=0;
   num_aviones=0;
   fuente1;fuente2;fuente3;fuente4;
   //fuente4;
   idprincipal;

   escrito1;
   escrito2;
   escrito3;
   escrito4;
   escrito5;
   escrito6;
   escrito7;
   escrito8;

   escriton; escritof; xf;
   dura_esc=0;

   //sonidos
   s_cohete;   s_salto2;   s_muerte;   s_voltaje;
   s_disparo;  s_bono;     s_d_malo;   s_llamear;
   s_bote;     s_lanza;    s_d_burro;  s_moto;
   s_granada;  s_toquemalo; s_chispa;  s_moto2;
   s_explosion; s_muertemalo; s_camicaze; s_franco;
   s_llamarada; s_venganza; s_diri;    s_mini;
   s_toque;    s_avion;    s_quejido;  s_tele;
   s_escudo;   s_camion;   s_choque;   s_fuma;
   s_salto;    s_bomba;    s_metal;    s_canto;

   s_recarga;  s_med_sal;  s_burbuja;  s_sirena;
   s_misiles;  s_a;        s_grana;    s_seco;
   s_casca;

   //m£sicas;
   s_todos;    s_game;     s_extra;
s_truco;
   s_muerto;   s_fin;      s_inicio;


local
   velocidad=0;
   velocidad_Newton;
   num_kamikazes=0;
   num_rasos=0;
   angulo4;
   angulo5;

begin
```







Fénix también funciona con numerosos juegos de DIV sin ningún tipo de cambio.

De esta forma creamos un nuevo fichero, en formato *bytecode*, que tendrá el mismo nombre que el fichero anterior pero con la extensión DCB. Si no se ha producido ningún error, ahora debemos ejecutarlo con el otro programa con la siguiente sintaxis:

FXI fichero.DCB

ya podremos disfrutar de Fénix en todo su esplendor.

Las novedades de Fénix con respecto a DIV

Fénix tiene gran cantidad de mejoras respecto a la versión 1 de DIV, muchas de ellas forman ya parte de la nueva versión de DIV, sin embargo otras son completamente nuevas. Veamos algunas de las más importantes:

1. Modos de 16 bits de color

Tal vez una de las más llamativas y novedosas es la utilización de gráficos de 16 bits en la ejecución de los programas, lo que nos permite olvidarnos de una vez por todas de las paletas de colores. Para ello existe una utilidad que se distribuye junto con Fénix llamada MAP. Este programa se encarga de convertir los archivos MAP de 8 bits de color a los nuevos archivos MAP de 16 bits que puede utilizar Fénix y además permite la creación de archivos PNG también de 16 bits y que podemos utilizar de forma indiferente junto con los archivos MAP.

Estos son los comandos que debemos escribir para utilizar esta aplicación:

- *map -l fichero.map:* muestra información del fichero MAP (nombre, dimensiones, bits, y puntos de control). Es la opción por defecto.
- *map -p fichero.map:* extrae la paleta de colores del fichero MAP (no es válido en MAPs de 16 bits). Le da el mismo nombre que al fichero original, y extensión .PAL
- *map -c fichero.map:* convierte un fichero MAP de 8 bits, a 16 bits. Sobrescribe el fichero original.
- *map -g fichero.map:* convierte un fichero MAP (tanto de 8 como de 16 bits) a PNG. Crea un fichero de mismo nombre y extensión .PNG
- *map -m fichero.png:* crea un fichero MAP (tanto de 8 como de 16 bits) a partir de un PNG. Crea un fichero de mismo nombre que el original y extensión .MAP

Todas las opciones admiten más de un fichero en la línea de comandos. Además, las opciones -l, -c y -m permiten editar mediante comandos adicionales, los puntos de control y el nombre del mapa. Estos comandos tienen el siguiente formato:

- *+name=nombre:* Asigna un nuevo nombre al fichero MAP.
- *+center=x,y:* Asigna la posición x,y (donde x e y son números enteros) como el centro del gráfico. Usar "+center=" para eliminar el centro de la tabla de puntos de control.
- *+n=x,y :* Asigna un punto de control, "n" es un número entre 0 y 999. El comando funciona igual que el anterior.

En el caso de que se estén listando, creando o convirtiendo múltiples ficheros MAP, éstos comandos afectarían a todos por igual.

Ahora sólo nos queda poder utilizar estos gráficos en nuestro juego. Para ello existe una variable global predefinida llamada *GRAPH_MODE*, cuyo valor por defecto es 0, que indica que tendremos gráficos de sólo 8 bits de color, pudiendo valer también la constante *MODE_16BITS*, lo que indica que podremos utilizar en nuestro juego tanto gráficos de 16 y 8 bits de color. Este valor sólo se puede utilizar al comienzo del programa y antes de que se cargue cualquier fichero o mapa. Es importante saber que no podremos utilizar efectos de paleta en imágenes de 16 bits de color, como fundidos y rotaciones.

Podemos observar cómo esta nueva variable global es un añadido a DIV, y que por tanto, al igual que otras muchas instrucciones nuevas, es incompatible con ambas versiones de DIV.

2. Tipos de dato

Otro añadido bastante importante es la inclusión de tipos de dato, que si bien puede complicar el código, también puede servirnos para realizar una gran variedad de operaciones que antes no podíamos hacer. Por tanto, cuando declaremos una variable, debemos indicar su tipo asociado. Esto se realizará de la forma:

TIPO variable;

Los tipos de datos de que disponemos son los siguientes:

- *BYTE:* se corresponde con un byte de memoria, y tiene un rango de valores de 0 a 255.
- *SHORT:* ocupa 2 bytes (16 bits) en memoria, y su rango de valores posibles va desde -32767 a 32767.
- *INT:* es un entero de 4 bytes (32 bits) y por tanto su rango de valores es de -2147483647 y 2147483647. Tanto con este tipo como con los anteriores, podemos realizar todo tipo de operaciones aritméticas, lógicas y de desplazamiento.
- *FLOAT:* es un número con coma flotante de 32 bits. Sobre este tipo de dato sólo se pueden utilizar operaciones aritméticas.
- *STRING:* permite manipular cadenas de caracteres. Su importancia radica en el uso de funciones de manipulación de cadenas, que deben recibir como parámetro una variable de este tipo.
- *TYPE:* permite crear nuevos tipos de dato a partir de los anteriores. Para ello debemos utilizar la siguiente sintaxis:

Fénix es un programa de distribución gratuita cuyo código puede ser manipulado por el usuario

TYPE nombre
--- variables ---
END

Deben declararse al comienzo del documento, preferiblemente antes de la sentencia *GLOBAL* o *LOCAL*. Posteriormente podremos utilizar estos tipos como si de un nuevo tipo de dato se tratara. Para acceder a sus elementos, la operación se realizará como en las estructuras. Esta es la sintaxis:

Nombre.variable

Lo único que no podremos realizar con estos tipos de dato es la asignación directa y la compara-

ción entre dos variables de este nuevo tipo. Para ello, debemos comparar los elementos del nuevo tipo uno por uno. Veamos un ejemplo:

```
TYPE coordenada
    SHORT x;
    SHORT y;
END

GLOBAL
    Coordenada coor1;
    Coordenada coor2;
BEGIN
    Coor1.x = 300;
    Coor1.y = 200;
    Coor2.x = 100;
    Coor2.y = 400;
    IF (Coor1 < Coor2 ) // ERROR
    IF (Coor1.x < Coor2.x)
//CORRECTO
        FRAME;
END
```

- *POINTER:* permite crear variables de tipo puntero, es decir, que contienen direcciones de memoria. Estas variables sólo podrán apuntar, es decir, contener la dirección de memoria, a una variable del tipo indicado en su declaración. Esto quiere decir tomando este ejemplo:

```
Int pointer puntero;
```

Que sólo podrá contener direcciones de memoria de variables de tipo INT. Para este tipo de variables, impera la sintaxis de C, que consiste en lo siguiente:

Fénix puede convertirse en una estupenda alternativa mientras esperamos DIV para Windows

Si queremos acceder al valor de la dirección de memoria de la variable, debemos anteponer el símbolo & al nombre de la variable. De esta forma, podemos asignar a punteros la dirección de otras variables de la siguiente forma:

```
Int pointer puntero;
Int valor = 20;
Puntero = &valor;
```

Por tanto, la variable *puntero*, "apunta" a la variable *valor*.

Si queremos acceder al valor contenido por la dirección de memoria que posee una variable de tipo puntero, debemos hacer lo siguiente:

```
Int valor2;
Valor2 = *puntero;
```

De esta forma, *valor2* tendrá asignado como *valor*, 20, que es el contenido de la variable valor, a la que apuntaba el puntero.

Los punteros nos permiten realizar gran cantidad de operaciones útiles, sobre todo en el recorrido de arrays y en la asignación de punteros de tipos definidos por nosotros. Este campo es muy amplio, y por tanto remitimos a la propia documentación de Fénix para obtener más información acerca de este tema.

3. Declaración de procesos

A diferencia de lo que ocurre en DIV y como consecuencia del uso de tipos de variable, la declaración de procesos varía con respecto a DIV, si bien esto viene a ser un añadido, funcionando correctamente el modo de procesos de DIV sin ningún tipo de variación.

En DIV colocábamos entre paréntesis los parámetros que la función recibía. Fénix permite declarar el tipo de dato de cada uno de los parámetros, siguiendo la siguiente sintaxis:

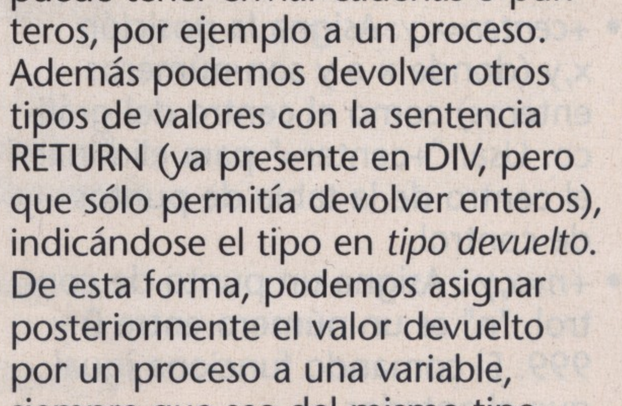
```
PROCESS <tipo devuelto> nombre (tipo1 parametro1, tipo2 parametro2,...)
```

De esta forma, podemos aprovecharnos de las ventajas que puede tener enviar cadenas o punteros, por ejemplo a un proceso. Además podemos devolver otros tipos de valores con la sentencia RETURN (ya presente en DIV, pero que sólo permitía devolver enteros), indicándose el tipo en *tipo devuelto*. De esta forma, podemos asignar posteriormente el valor devuelto por un proceso a una variable, siempre que sea del mismo tipo que el del valor devuelto. Si no existe una congruencia entre ambos, Fénix no lo detectará, con lo que hay que tener cuidado con ello.

Es importante tener en cuenta que la ausencia de cualquier tipo de dato implicará considerar el parámetro o el tipo de dato correspondiente como un tipo INT, como es considerado en DIV.

4. Inclusión de programas externos

Fénix nos permite escribir programas muy largos y dividirlos en módulos separados para, de esta forma, beneficiarnos de la distribución del código. Así, la legibilidad será mucho mayor, y además podremos movernos con mayor facilidad por el código. Aunque esta opción no está disponible en DIV, existe un programa que se encarga de enlazar el código de estos archivos en uno solo más largo.



Fénix incluye gran cantidad de programas de ejemplo que permiten probar todas sus capacidades.

Esta unión de programas se realiza mediante la sentencia *INCLUDE* de un modo bastante parecido a como lo hacen otros compiladores, como C o C++. Sin embargo, no existe el concepto de modularidad y privacidad de los diferentes módulos, sino que es una separación de conglomerados de código. Es decir, el código de los módulos así incluidos se añade al código del módulo principal desde el que se incluyen. La sintaxis que debemos seguir es la siguiente:

```
INCLUDE "fichero"
```

Debemos colocar esta sentencia al comienzo del programa, antes de cualquier declaración de cualquier tipo. Dentro de todo el entramado de procesos resultantes, pueden existir entradas al código principal del programa, que se distinguirá por la presencia de las líneas *BEGIN* y *END* sin ninguna sentencia del tipo *PROCESS*. La existencia de dos o más casos de este tipo provocará que cada una de las entradas al proceso principal se ejecuten por orden de inclusión en el programa. Esto quiere decir que, en primer lugar, se ejecutará el código principal del fichero principal y después se ejecutarán todos los códigos principales, según el orden en que se hayan incluido en el programa principal con la sentencia *INCLUDE*.

Tanto las variables globales como locales, tipos de dato y procesos de otros módulos serán accesibles desde cualquier punto del programa, debiendo tener el debido cuidado para no caer en el error de declarar dos veces el mismo tipo.

5. Nuevas funciones

Además de estas novedades, Fénix incluye una serie de nuevas funciones de las que, por su diversidad, sólo analizaremos las más importantes:

- *Funciones de carga y creación de imágenes:* además de añadir los formatos PCX y PNG al formato MAP de imágenes, podemos realizar la carga de imágenes con 16 bits de color. Para ello podemos utilizar las funciones:

Int LOAD_PNG (string nombre);
Int LOAD_PCX (string nombre);
Int NEW_MAP (int ancho, int algo, int bitsdecolor);

Al igual que ocurre con la función *LOAD_MAP* de DIV, la función devuelve un identificador del gráfico que podremos utilizar posteriormente para su asignación a procesos. El número del fichero al que pertenecen estas imágenes, se considera 0, al igual que en DIV. La instrucción *NEW_MAP* incluye un tercer parámetro, *bitsdecolor*, que indica los bits (8 ó 16) que se utilizarán en el nuevo mapa creado.

- *Control de sonido:* aunque no se encontraba disponible la reproducción de archivos MOD, S3M, IT o XM en DIV, esto ya apareció en la segunda versión. Por ello también lo hace Fénix, que implementa de forma parecida la posibilidad de reproducción de estos formatos mediante las instrucciones *LOAD_MOD(string nombre)*, *PLAY_MOD(int indice)* y *STOP_MOD()*, que reciben, la primera, el nombre del fichero a reproducir, devolviendo un índice que utilizaremos en la segunda función para reproducir dicho archivo. Para parar la reproducción de cualquier archivo, utilizaremos la última función.

El programa Fénix está todavía en versión beta a falta de que se acabe una versión definitiva

- *Gestión de cadenas:* existen numerosas funciones para la manipulación de cadenas, así como para la conversión de tipos que, por su extensión, sólo describiremos algunas como:
  - *String SUBSTR(string cadena, int inicio, int fin):* devuelve una cadena que contiene los caracteres de la cadena pasada como parámetro que se encuentran en la posición inicio hasta fin. Los números negativos en los índices, se consideran como posiciones a partir de la última letra de la cadena.
  - *Int LEN (string cadena):* devuelve un entero con la longitud de la cadena especificada.
  - *String ITOA (int entero):* devuelve una cadena que contiene el número pasado como parámetro.
  - *Int ATOI (string cadena):* devuelve el entero correspondiente a una cadena. Si no es correcta la cadena, devuelve 0.
  - *Float ATOF (string cadena):* devuelve el valor de tipo float correspondiente a la cadena. Si no es correcta, devuelve 0.0;

Y esto no es todo

Es evidente que un compilador como DIV no se puede resumir en unas pocas páginas, y ése es también el caso de Fénix. Nos hemos dejado muchas cosas en el tintero que nos gustaría relatar.
Pretendemos que ésta sea una presentación de un producto que, sin duda, puede ser el futuro de DIV y que, a través de su conocimiento, la calidad de los programas mejore.

Nada más que animar a José Luis Cebrián a continuar con su proyecto, así como felicitarle por su maravilloso trabajo. Para cualquier información, crítica o pregunta, pueden dirigirse como siempre a trinidad@arrakis.es

Pablo Trinidad

Fénix en nuestro Cd

En el Cd que acompaña a Divmanía podéis encontrar Fénix, en sus versiones para Windows y Linux, y un juego de ejemplo, *La rata y las serpientes*. Como es lógico encontraréis también toda la información necesaria para su instalación y uso que os resumimos aquí a través de las palabras del propio autor:

¿Qué es Fenix?

Fenix es un lenguaje pseudo-interpretado, diseñado para poder hacer juegos de dos dimensiones (a base de "sprites") fácilmente. Para ello consta con una extensa librería incluída, dedicada a los juegos, que permite programar únicamente la lógica de movimiento de los gráficos y objetos del juego, mientras cosas como la visualización en pantalla de gráficos y sprites, o la reproducción del sonido, corren a cargo del intérprete incluído.

El lenguaje es un Pascal reducido, con algunas diferencias notables e influencias de otros lenguajes. Es relativamente compatible con DIV, un sistema de desarrollo de videojuegos publicado por Hammer Technologies. Los juegos hechos en DIV pueden convertirse con poco esfuerzo a Fenix, e incluso un cierto número de juegos de demostración de DIV se compilan y ejecutan sin problemas en Fenix, o con problemas menores. Todo ello se refiere a la versión 1 de DIV, ya que el lenguaje de Fenix es muy diferente de la versión 2 de este sistema.

¿Cómo usarlo?

Fenix consta de dos ejecutables:
– El compilador, FXC, lee el código fuente de un programa (.PRG) y escribe un fichero en formato binario "bytecode" (.DCB).
– El intérprete, FXI, lee un fichero en formato binario (.DCB) y ejecuta el programa en sí.

Un fichero DCB no es una mera conversión de los programas a otro formato, sino una especie de "lenguaje ensamblador" que entiende el intérprete. Puedes usar libremente los ficheros DCB resultantes de compilar tu programa.

Además, el fichero DCB ha sido diseñado para que sea compatible entre distintas versiones del intérprete. Si distribuyes tu fichero DCB, alguien que disponga de un intérprete para otra plataforma podrá ejecutarlo; y si en el futuro surge un intérprete más potente o más rápido, podrás ejecutar tus antiguos DCB con el nuevo intérprete y aprovechar las mejoras.

Debo hacer notar que el formato DCB aún no está acabado, y es probable que no lo esté hasta la versión 1.0. Desaconsejo distribuir ficheros DCB hasta que dicha versión sea una realidad, ya que los DCB actuales no funcionarán con intérpretes futuros.

Para compilar un fichero PRG bastan con ejecutar “FXC fichero.prg”

Si no se obtiene ningún mensaje de error, FXC habrá creado un fichero "fichero.dcb" en el mismo directorio, y ofrecerá algunas estadísticas sobre lo que contiene.

El siguiente paso es ejecutarlo con “FXI fichero.dcb”

Si el programa tiene problemas al abrir ficheros FPG, MAP y demás, ten presente que FXI cambia al directorio del DCB cuando lo abre. Así, si el programa quiere abrir el fichero "FPG/TEST.FPG", ese fichero deberá estar en el subdirectorio FPG del mismo directorio donde esté el DCB para funcionar.

Si FXI no logra abrir un fichero, hace un segundo intento buscando un subdirectorio en el directorio del DCB que tenga el mismo nombre que la extensión del fichero. Así, si no encuentra el fichero "TEST.FPG", probará a abrirlo como "FPG/TEST.FPG".

Open GL

Un estándar para manejar gráficos

La creciente utilización del ordenador en el ámbito doméstico/lúdico, en la arquitectura y en la industria cinematográfica, requería un estándar de utilización y programación potente, rápido y asequible para visualizar en tiempo real una gran cantidad de polígonos y gráficos bidimensionales.

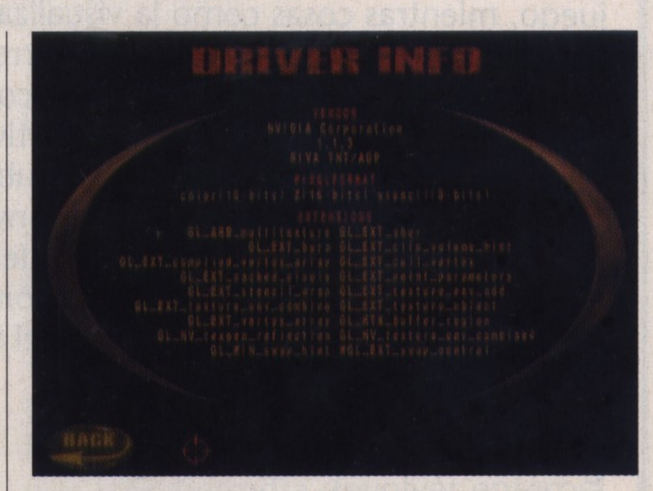
A principios de los años noventa se produjo un "boom" de tarjetas gráficas aceleradoras 2D/3D. Cada una era incompatible con las otras, tenían *chipsets* distintos y funcionaban de diferente manera, unas permitían hacer cosas que no podían realizar las otras. A medida que pasaba el tiempo iban incorporando más y más novedades tecnológicas. Fue entonces cuando la prestigiosa compañía Silicon Graphics decidió crear su Open Graphics Library.

¿Qué es OpenGL?

Se trata de una interfaz para manejar gráficos en dos y tres dimensiones utilizando, si es posible, todas las cualidades que nos ofrezca nuestra tarjeta gráfica y microprocesador. Para ello, especifica una serie de funciones y procedimientos que todo *driver* OpenGL debe soportar obligatoriamente (a esto lo llama *invariance* y, por ejemplo, es el pintado de polígonos, creación de determinados tipos de luz y operaciones sobre el *framebuffer*) junto con otros que son opcionales.

La batalla

Todo iba increíblemente bien para Silicon Graphics: veía cómo sus estaciones de trabajo (Indy, Onyx, etc.) funcionaban con él a las mil maravillas; la todopoderosa Microsoft incluía OpenGL en su sistema operativo Windows NT, utilizándolo también en SoftImage; personas como John Carmack (sí, el del *Quake*) le prestaban todo su apoyo... Pero, de pronto, Microsoft

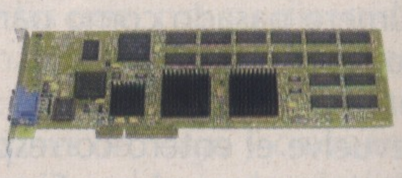


Aquí podemos ver cómo *Quake III* lista las *extensions* de OpenGL que se encuentran disponibles en el sistema.

se dio cuenta de que sus Windows 3.1 no eran "todo lo multimedia" que debieran serlo (ciertamente, programar juegos medianamente espectaculares con ellas era un calvario) y creó las Windows 95 con "una cosa de serie" llamada DirectX. Bajo este intrigante nombre se escondía una API para el manejo de sonido, dispositivos de entrada como teclado o joystick, gráficos 2D/3D, juego en red... que, poco a poco, fue comiendo terreno en el mercado lúdico del PC. Pero entonces ¿por qué incluir soporte para OpenGL en las Windows NT? Muy sencillo, porque este sistema operativo estaba destinado a uso profesional, donde existían multitud de programas CAD/CAM e infografía que no daban soporte a DirectX, destinado, más bien, al mundo de los juegos. Por si fuera poca la competencia en este sector, otra compañía, llamada 3Dfx, sacaba al mercado la revolucionaria tarjeta gráfica Voodoo y, con ella, su propia API, Glide.

Tanto OpenGL, DirectX y Glide (hay muchas más), sirven más o menos para lo mismo: para programar y utilizar gráficos en 2D/3D de forma relativamente sencilla. Todas tienen cosas buenas y malas, ventajas y desventajas sobre las otras. Elegir es una cuestión personal. Si preguntásemos a "Mr.Quake" defendería a muerte OpenGL porque, probablemente, era la opción más razonable en el momento de crear *Quake II* y, sin duda, ofrecía multitud de ventajas sobre el inmaduro DirectX que, no olvidemos, hasta su versión más reciente no permitía transformar o iluminar los polígonos mediante la tarjeta gráfica, mientras que OpenGL ya lo hacía desde su primera versión. Una operación matemática tan sencilla como la transformación de vértices debe ser optimizada lo máximo posible para proporcionar una adecuada velocidad a los *engines* utilizados en los juegos tridimensionales. Hasta la llegada de la fantástica GeForce los juegos de ordenador utilizaban la CPU para realizar estos cálculos. Además, hasta la séptima versión de DirectX no era posible esto, cosa que sí se permitía bajo OpenGL.

Para los fanáticos de 3Dfx, el mejor es Glide, porque se adapta mejor que ninguno al chipset Voodoo y a sus características. Y, en fin, las DirectX están excelentemente documentadas y las avala uno de los grandes de la informática. Pero, no nos engañemos, OpenGL tiene más tradición en el mercado profesional, es posible verlo en multitud de sistemas operativos, incluso en simuladores de



Aquí podemos observar una tarjeta gráfica profesional equipada con 96Mb de memoria de vídeo y dos potentes coprocesadores matemáticos serie Glint.

realidad virtual o vuelo real. Toda tarjeta aceleradora que presuma ser profesional debe contar con drivers para él y, sobre todo, tiene una arquitectura increíblemente abierta. De hecho, está tan bien pensado que la versión original sólo ha mutado dos veces, frente a las tres de Glide o las siete de DirectX.

Arquitectura

Utiliza procedimientos y funciones, lo cual lo hace fácil de usar, pero no resulta tan elegante y eficaz como la programación por objetos. Además, contiene un sistema de *extensions* (plug-ins) que permite actualizarlo sin necesidad de sacar una nueva versión. Debemos indicar que existen dos tipos de extensiones: las que son homologadas y revisadas por la propia Silicon Graphics (*bump, multitexture*) y otras que son propias de cada fabricante (por ejemplo, la referente a la compresión de texturas *S3TC* o los *cube map* de la nVidia GeForce que podemos ver en la foto).

Por supuesto, podemos listar todas las extensiones para saber cuáles están disponibles y utilizarlas para mejorar la apariencia gráfica o el rendimiento.

Con el fin de adaptarlo a diferentes sistemas operativos, se crean métodos específicos propios de cada sistema operativo. Por ejemplo, para los que hayan programado el GDI de Windows, podemos decir que es necesario crear un *device context* especial con el que funciona OpenGL, ya que necesita un formato de píxel característico. Por este motivo, se incluyó una función llamada *wglCreateContext* en las Windows 95 OSR2.

OpenGL cuenta con una arquitectura distribuida que reparte inteligentemente la carga de trabajo entre el/los microprocesadores y la tarjeta gráfica que, además, puede llevar a cabo la transformación, iluminado y *clipping* (el tan traído y llevado T&L de la nueva tarjeta gráfica de nVidia y que se lleva realizando hace años por costosísimas tarjetas con varias FPU integradas en ella, como las que utilizaban varios micros Glint) totalmente por hardware.

Utilización

Para el usuario resulta casi transparente: debe limitarse a instalar los drivers que le proporciona su tarjeta gráfica para disfrutar de su programa favorito. En cambio, el programador que desee incorporar OpenGL a su proyecto debe conseguir los ficheros de definiciones que le permitan conocer la sintaxis y parámetros de todas las funciones y procedimientos que pone a su disposición. En el caso de programar en lenguaje C, nos referimos, por supuesto, a los .H y .LIB. Estos archivos son fáciles de obtener, por ejemplo, en Internet (www.OpenGL.org) o, incluso, en los disquetes de *drivers* que acompañan a ciertas tarjetas gráficas.

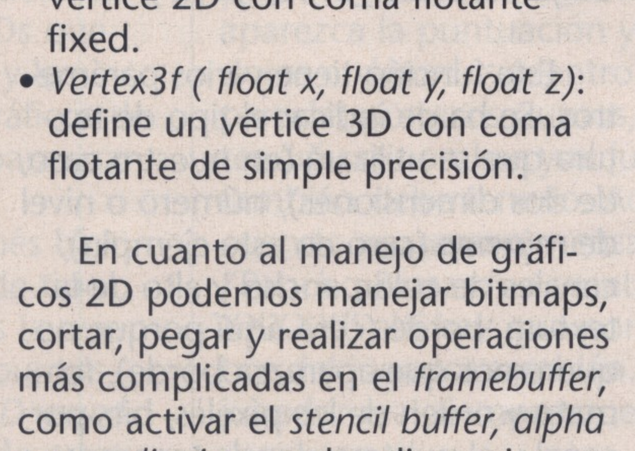
Además, podemos obtener abundante información en la Red sobre temas relacionados con él, tutoriales, foros de debate, ejemplos, etc... Basta con buscar la palabra "OpenGL" en cualquier buscador para encontrar miles de referencias. Y, al igual que en DirectX, con su D3Dframe, disponemos de una serie de librerías que sirven para hacerle la vida más fácil al programador. Nos referimos a GLAux y GLU o MESA, que no son más que un conjunto de procedimientos "extra" que permiten generar con una sola llamada esferas, torus, cargar texturas en formato SGI-RGBA, etc.

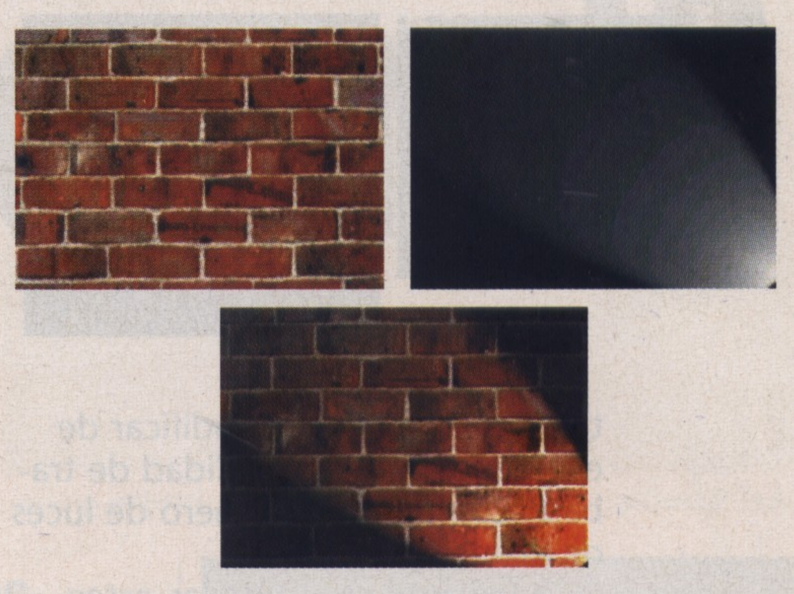
Entrando en materia

La mayoría de funciones se pueden utilizar con varios tipos de datos. Por ejemplo, podemos especificar las coordenadas de un vértice en coma flotante de simple o doble precisión, o enteros *fixed* en dos o tres dimensiones.

- *Vertex2i (int x, int y)*: define un vértice 2D con coma flotante fixed.
- *Vertex3f (float x, float y, float z)*: define un vértice 3D con coma flotante de simple precisión.

En cuanto al manejo de gráficos 2D podemos manejar bitmaps, cortar, pegar y realizar operaciones más complicadas en el *framebuffer*, como activar el *stencil buffer, alpha test* o *clipping* en dos dimensiones por *scissor test*, pasando por el típico *dithering* y *alpha blending* hasta llegar a las operaciones lógicas del tipo NOT, XOR. Además, contamos con multitud de filtros, incluyendo el de *convolución* para detalle del escalado, fundidos de color francamente complicados mediante *color histograms* y todo ello, por supuesto, aderezado con la posibilidad de





Aquí podemos ver un ejemplo de cómo se simula una luz mediante la utilización de dos texturas en tiempo real, la denominada técnica *multitexture*.

trabajar en diversos modos gráficos y formatos de píxel acelerados hasta la última gota por hardware.

Llegado el turno de los gráficos en tres dimensiones, ¿qué podemos decir? Pues nada más y nada menos que disponemos de todo tipo de vértices, desde no iluminados ni transformados con ningún tipo de material hasta los más procesados, por si queremos implementar nuestro propio sistema de T & L. Podemos utilizar el clásico método de generar polígonos por listas de triángulos, fans, quad, puntos, líneas o, si lo preferimos, podemos darle todos los vértices para que genere un polígono convexo o precompilar una lista para optimizar de forma drástica la malla. Utilizando librerías externas (y una potentísima tarjeta aceleradora) incluso es posible trabajar directamente con NURBS.

Open GL es una interfaz para manejar gráficos en dos y tres dimensiones

En cuanto al funcionamiento general es muy parecido a Glide o Direct3D: antes de comenzar a enviar datos debemos indicar a la tarjeta gráfica, mediante los respectivos comandos "Begin/End", que se prepare para renderizar. Dentro de estas llamadas podemos indicar las *matrices de transformación* oportunas para la rotación, posición y escalado de la malla, así como de la cámara o generar proyecciones ortogonales, coordenadas de textura automáticas, activar filtros bi/trilineales, *antialiasing* o *mipmapping*, activar trasparencias, asignar planos de *clipping* totalmente definibles o, incluso, obtener en cada momento la posición de rastering actual, realizar *mirroring / wrapping* sobre texturas.

Uno de los puntos fuertes es la calidad de imagen obtenida, la buena iluminación que se consigue a través de la infinidad de paráme-

tros que se pueden modificar de ella, así como la posibilidad de trabajar con un gran número de luces de varios tipos.

Mediante el uso de las *extensions* (mencionadas más arriba), podemos utilizar *lightmaps, bumping* o *anvironment cube maps* para dar mayor realismo a la escena.

Un pequeño ejemplo

Vamos a pintar un cuadrado texturizado para ver lo fácil que resulta programar con OpenGL utilizando lenguaje C bajo Windows.

1) Para utilizar las funciones que nos proporciona OpenGL, debemos incluir los archivos de cabecera *<gl.h>* y *<glaux.h>*. En Microsoft Visual C++ pondríamos:

OpenGL, DirectX y Glide sirven más o menos para lo mismo: programar y utilizar gráficos en 2D/3D

```
#include <GL/gl.h>
#include <GL/glaux.h>
```

2) Ahora, inicializamos la ventana donde pintaremos el triángulo mediante las órdenes:

```
auxInitDisplayMode (AUX_SINGLE | AUX_RGB);
auxInitPosition (0, 0, 640, 480);
auxInitWindow ("Mi aplicacíon en OpenGL");
```

La primera de ellas indica que se utilizará un *single buffer* (lo conveniente sería utilizar, al menos, *doble buffer* para evitar parpadeos al pintar) en modo RGB. La segunda y tercera muestran una ventana de 640x480 pixeles en las coordenadas de pantalla 0,0 (esquina superior izquierda), visualizando en su cabecera el título " Mi aplicación en OpenGL ".

- ARB – Extensions officially approved by the OpenGL Architectural Review Board
- EXT – Extensions agreed upon by multiple OpenGL vendors
- HP – Hewlett-Packard
- IBM – International Business Machines
- KTX – Kinetix, maker of 3D Studio Max
- INTEL – Intel
- NV – NVIDIA Corporation, coolest 3D company on the planet
- MESA – Brian Paul's freeware portable OpenGL implementation
- SGI – Silicon Graphics
- SGIX – Silicon Graphics (experimental)
- SUN – Sun Microsystems
- WIN – Microsoft

Cada *extension* dispone de tres siglas que indican su fabricante.

3) El paso siguiente es borrar el fondo de la ventana con color negro.

```
glClearColor (0.0f, 0.0f, 0.0f, 0.0f );
glClear (GL_COLOR_BUFFER_BIT);
```

La primera línea establece un color RGB negro para el borrado. La segunda borra el fondo de la ventana.

4) Ahora llega el momento de cargar la textura. Para ello podemos utilizar un cargador PCX / TGA o leer desde un archivo .SGI los valores RGBA en bruto. Una vez hecho esto debemos activar el texturizado en dos dimensiones mediante

```
glEnable (GL_TEXTURE_2D);
```

Existe la posibilidad de trabajar con texturas en tres dimensiones para simular efectos de relieve o *bumping*, pero para simplificar este ejemplo lo haremos así.

A continuación, seleccionamos la textura con la que pintar nuestro cuadrado mediante:

```
glTexImage2D ( GL_TEXTURE_2D, 0, 4, 64, 128, 0, GL_RGBA, GL_UNSIGNED_BYTE, lpTex );
```

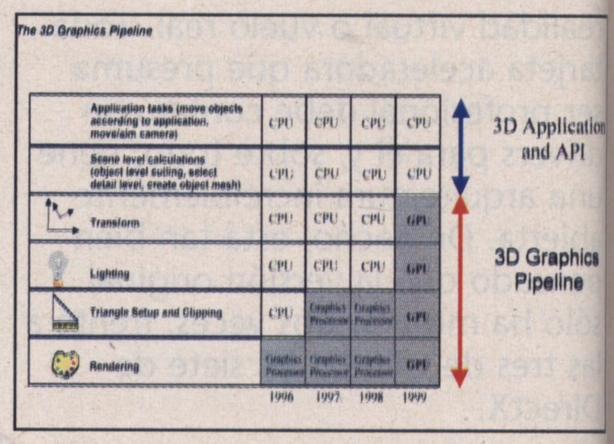
Esta función tiene varios parámetros. Se ha de indicar el tipo de textura que se utilizará (en nuestro caso, de dos dimensiones), número o nivel de *mipmap* (cero en este ejemplo), canales de color, ancho y alto de la textura, borde (cero aquí porque no queremos que aparezca borde), formato y orden de los píxeles, bits por canal y el puntero donde tenemos los píxeles de la textura.

Podemos activar el *filtro bilineal* para mejorar la calidad de imagen mediante:

```
glTexParameterf (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterf (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

Indicando que se utilizará a la hora de aumentar y disminuir relativamente la coordenada Z del polígono que la utilice.

También podemos realizar efectos de *mirroring* o desactivarlos. En este último caso utilizamos:




```
glTexParameterf (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameterf (GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
```

para no permitir que las coordenadas de textura se encuentren en todo momento entre 0.0f y 1.0f.

5) Ahora pintamos un rectángulo formado por dos triángulos. Para ello es imprescindible hacerlo entre las funciones *glBegin* y *glEnd*.

```
glBegin (GL_TRIANGLES);
    glTexCoord2f (0.0f, 1.0f);
    glVertex3f (-0.5f, 0.0f, 0.0f);

    glTexCoord2f (1.0f, 1.0f);
    glVertex3f (0.5f, 0.0f, 0.0f);

    glTexCoord2f (1.0f, 0.0f);
    glVertex3f (0.5f, 0.5f, 0.0f);

    glTexCoord2f (0.0f, 1.0f);
    glVertex3f (-0.5f, 0.0f, 0.0f);

    glTexCoord2f (0.0f, 0.0f);
    glVertex3f (-0.5f, 0.5f, 0.0f);

    glTexCoord2f (1.0f, 0.0f);
    glVertex3f (0.5f, 0.5f, 0.0f);

glEnd ();
```

Obsérvese que las coordenadas de textura corresponden al mapeado de la figura 12.

Conclusiones

A pesar de lo básico de este ejemplo y las matizaciones que cabría hacer de él, podemos observar los mecanismos básicos de funcionamiento y lo fácil que resulta hacer las cosas en comparación con otras APIs para el manejo de gráficos 2D / 3D. En próximas entregas iremos descubriendo todas las posibilidades que nos brinda OpenGL.

Santiago Orgaz Sánchez
Santyhammer@latinmail.com

Curso de juegos en 3D

DIV Deathmaker (III)

Tercera entrega de nuestro curso de juegos en tres dimensiones. En este número veremos a fondo cómo funciona nuestro gestor de bots y en general, cómo vamos a manejarnos con los datos de cada bot.

Bueno, ya tenemos el juego completamente planeado. ¿Y ahora qué? Parece que va siendo hora de entrar profundamente en programación. Pues no hay miedo, allá vamos.

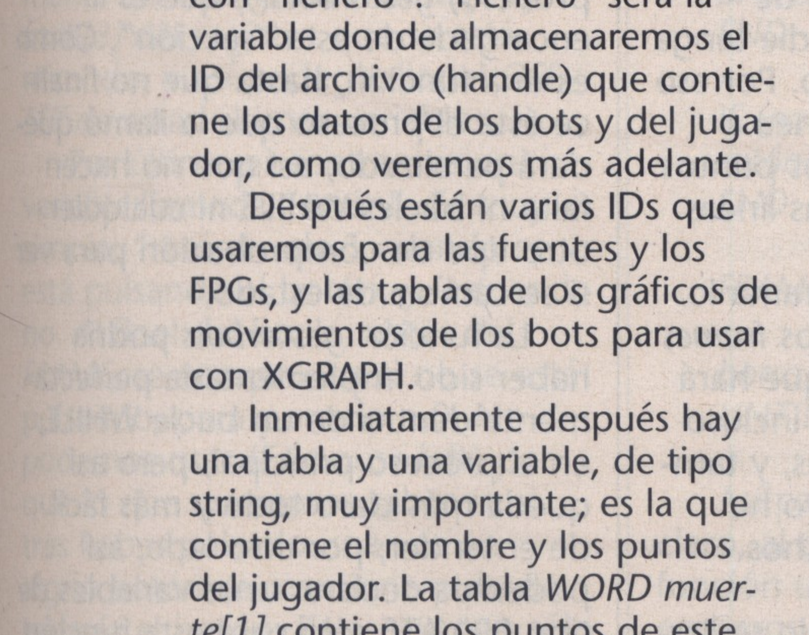
Las variables

Las variables del PRG del anterior número han sido modificadas y varias más añadidas. Aquí explico un poco su uso.

Nada más mirar un poco por encima nos encontramos con la declaración de variables globales y con "fichero". "Fichero" será la variable donde almacenaremos el ID del archivo (handle) que contiene los datos de los bots y del jugador, como veremos más adelante.

Después están varios IDs que usaremos para las fuentes y los FPGs, y las tablas de los gráficos de movimientos de los bots para usar con XGRAPH.

Inmediatamente después hay una tabla y una variable, de tipo string, muy importante; es la que contiene el nombre y los puntos del jugador. La tabla *WORD muerte[1]* contiene los puntos de este modo: *muerte[0]* contiene el número de asesinatos totales del jugador, y *muerte[1]* contiene el número de muertes totales. Así, la puntuación neta se obtiene a partir de la resta del primero menos el segundo. Para complementar a ésta, la variable *STRING nombre* contiene el nombre del jugador, de hasta 15 caracteres, al igual que los de los bots. Por último, la razón de que la tabla sea de WORD es obvia: al ser tablas se pueden aprovechar los datos tipo BYTE y WORD, y como son dos, preferiblemente de mayor valor que 256, pues 2*2 bytes cada word=4, o sea, una alineación de memoria.



El proceso de creación del mapa DEFAULT.WLD no fue precisamente corto.

Quizá pensáis que el ponerle nombre al jugador sea algo inútil porque no hay que identificarlo, pero no es así. Cuando en pantalla aparezca la puntuación y los jugadores ordenados (dentro de una partida) por esos puntos, cada uno tendrá un nombre y el jugador también debe tenerlo. Además, cuando aparezcan mensajes tipo "Pancho Brasas fue ametrallado por XXXXX" ahí hay que poner el nombre del jugador también... y así el jugador real siempre puede identificarse con él.

Pasamos a la estructura *bot[29]*, modificada con respecto a la del anterior número. Como podréis observar, tenemos una tabla y una variable nuevas, éstas son la tabla *WORD muerte[1]*, exactamente igual que la del jugador pero aplicable a los bots, y una variable que os podría parecer inútil, pero al revés, es muy importante, llamada puntos.

Cuando queramos elegir los bots que van a jugar, éstos van a aparecer en 30 filas, justo debajo del nombre del jugador, ordenados de MÁS a MENOS puntuación neta, mostrando la de cada uno (y la del jugador) a la derecha. Si pensáis cómo vamos a poder ordenarlos, y

Hete aquí algunos de los gráficos de nuestro juego (MISC.FPG).

os viene a la cabeza una tonelada de IFs calculando todas las posibilidades combinatorias... pues lo mejor es usar la función *QSORT*, que ordena los registros de una estructura en relación al orden de un campo. Si nosotros rellenamos, en cada registro, la variable puntos con la puntuación neta de cada bot, podemos usar *QSORT* para ordenarlos en orden descendente dependiendo de este campo. Por ello, más adelante nos será muy útil.

Llegando a la estructura *player[7]*, ésta es una estructura importante, también. Cuando se comienza un nuevo juego, tras elegir los bots, el escenario, etc... el jugador y los bots tienen su puntuación propia y hay que ordenarlos en tiempo real, en la partida, para mostrar el ranking de la partida. No podemos usar *QSORT* si no están todos los registros en una misma estructura (los datos del jugador no están en ninguna) y, además, no nos conviene modificar directamente la puntuación en los registros de *bot[29]* o los del jugador, así que esta estructura nos vendrá de perlas para almacenar temporalmente todos esos datos.

Dentro hay una variable "who" que indica a qué registro de *bot[29]* pertenecen los datos almacenados en ese registro de *player[7]*; esto es necesario dado que tras muchos *QSORTs*, el que estuviera en *player[1]* puede acabar en *player[6]*, y no habría modo de identificarlos. Para indicar que es el registro del jugador, *who* debe valer 30. A la hora de acabar la partida (porque se acabe el tiempo o se

Así queda, de momento, nuestro menú principal.

alcance el fraglimit) who indicará a qué bot debe devolver cada registro de los datos.

También está la ya conocida tabla WORD *muerte[1]*, y la *STRING nombre*. Las restantes que se utilizarán en medio de la partida (los atributos) son propias de cada proceso de bot, y no se modifican durante el juego, así que no necesitan ser incluidas aquí.

Por último dentro de la sección GLOBAL, está la tabla WORD *limit[1]*, que contiene los límites impuestos por el jugador para la partida. *Limit[0]* indica el límite de tiempo (time limit) en minutos, una vez transcurrido este tiempo se acabará la partida. Y *limit[1]* contiene el fraglimit que hay que alcanzar para que acabe la partida, pero atención, no en número de asesinatos (frags) sino en PUNTUACIÓN NETA. Si alguno de estos dos registros de la tabla vale 0, indica que no hay límite de ese tipo, y por supuesto, se pueden poner los dos tipos de límite o, en caso contrario, no poner ninguno, pero hay que tener en cuenta que si no hay límite la partida nunca acaba, con lo que no hay resultado final, y esto implica que no se actualizan las puntuaciones de los bots ni del jugador, ya que al parar la partida (interrumpirla para salir, pulsando ESC) no se guardan las puntuaciones. Se podría considerar entonces que este modo de juego es de tipo de "entrenamiento".

Pasando a las variables locales, es decir, las propias de cada proceso, encontramos unas pocas. "Arma" indica qué arma está activa y de qué modo. Es decir, qué número de arma, y si está preparada, o cargando, o la está sacando, etcétera. Cada arma dispone de una decena para esto, es decir, el arma 1 (láser de impulsos) activa y preparada sería un valor 10; 11 sería si es el láser de impulsos, pero se está seleccionando (sacando) o se está guardando; 12 si se está cargando o se está preparando para el siguiente disparo, y así con la siguiente decena para la Qk22, etcétera.

La tabla WORD *hp[1]* contiene los puntos de salud y de blindaje del bot o jugador. El registro *[0]* contiene la vida, de máximo 200, y *[1]* los puntos de blindaje, de máximo también 200.

La tabla WORD *municion[3]* guarda la munición de cada tipo, de este modo: el registro *[0]* contiene la munición de balas, es decir, válidas para la Qk22 y para la ADC, con un máximo de 150. Cuando alcanza un valor divisible por 10 (tras disminuir), toca recargar. El registro *municion[1]* almacena el número de células de energía, teniendo un máximo posible de 300 (sólo por este registro, que sobrepasa los 256, ya no nos valen los datos de tipo BYTE). El registro *[2]* contiene el número de misiles, máximo 20, y [3] las minas, pudiendo tener hasta 10.

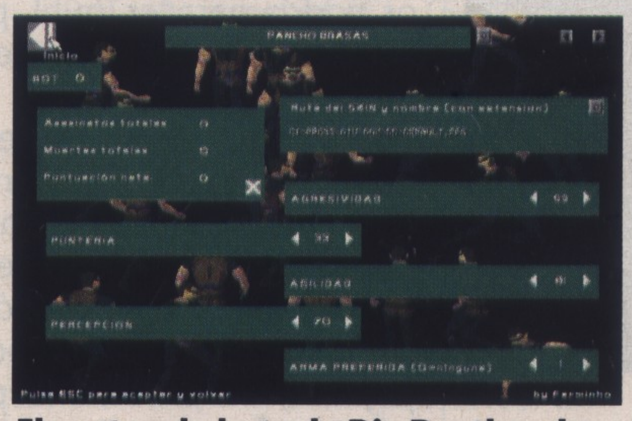
Y ya por último, está la tabla WORD *arma[3]*, que simplemente indica qué armas se tienen y cuáles no. (1=tiene; 0=no la tiene). Como siempre se empieza con el láser de impulsos y la Qk22, y como las minas no requieren de un arma para usarlas, sólo hay 4 registros: para la ADC, el bláster de fusión, el repetidor láser y el lanzamisiles.

Entrando en el código

Todo el código está bien comentado, y todo muy explicado, de modo que no creo que nadie tenga problemas para entenderlo. Por eso no voy a explicarlo todo línea a línea, pero si tocaré más los puntos más importantes, y en unas líneas más generales.

Tras el BEGIN del programa lo primero que hace es fijar los *frames per second (fps)* a 100, lo que hará que vaya bastante suave e incluso rápido en equipos potentes, y también poner la resolución (yo he optado por 640x400, muchos sabrán por qué ;).

Una vez se pone el título se cargan las fuentes, e inmediatamente después abre el fichero BOTS.CFG para cargar los datos de bots/jugador y lo cierra. En cuanto se descarga el título se aprovecha para FORZAR la paleta, con lo que ya todos los gráficos que se carguen quedarán adaptados a esa paleta, y



El gestor de bots de Div Deathmaker.

comienza el bucle que ya expliqué en el número pasado. Este era el bucle encargado de mirar dónde se encontraba el jugador en cada momento, que ahora veréis que está ampliado. Antes sólo actuaba si el estado era 0, es decir, si se estaba en el menú principal. El bucle WHILE lo mantenía en esa condición *IF (estado==0)* hasta que el valor de "estado" cambiaba.

Ahora también se puede acceder al gestor de bots (opción 2-BOTS), cuyo valor de "estado" es 2, y se puede retornar también de éste. Cuando se sale del estado 0 se descarga todo lo que ya no es necesario para los demás "estados", como los textos y el título, y se borra la pantalla. ¿Y qué pasa cuando el estado es igual a 2? Vamos a verlo con más detenimiento.

El gestor de bots

Cuando pulsamos 2 en el menú principal aparece una nueva pantalla, es la pantalla del gestor de bots. Echémosle un vistazo al código. Este apartado empieza con el "IF (estado==2)".

Lo primero que hace es poner de fondo la pantalla del gestor de bots (gráfico 999 en MISC.FPG), y justo después digamos que se "desentiende" y llama a la FUNCIÓN (ojo, función, que no proceso) *gestorbots()*, que es la encargada de esta "sección". Como es una función, hasta que no finalice ésta el proceso que la llamó quedará paralizado, así que no hacen falta ni bucles WHILE ni cualquier otro tipo de comprobación para ver si se cambia de estado.

La función *gestorbots* podría haber sido implementada perfectamente dentro de un bucle WHILE, en el proceso principal, pero así queda más claro todo y más fácil de entender; por otro lado, así podemos declarar unas variables de tipo PRIVATE sólo para esta función.

La función *gestorbots()*

Las variables privadas son esenciales. Primero, no vienen definidas en la sección PRIVATE, las variables X e Y. Sí, ya sé que no son nuevas para vosotros. Nos servirán como contadores. Pensad que, como no tienen utilidad, ¿para qué declarar otras nuevas si ya tenemos, obligatoriamente, éstas definidas? Ahorremos memoria, aunque no sea mucha.

X nos dirá en qué registro de *bot[29]* estamos. Si X vale -1 entonces es que estamos en el registro (imaginario) del jugador. Esta numeración viene representada en la parte superior izquierda, donde pone "BOT". Si estamos en el juga-

dor, pondremos que salga una P en vez de "-1".

Y sólo nos servirá como variable temporal a la hora de ordenar con QSORT los registros, para un bucle *FROM*. Lo normal sería *FROM n=1 TO 100* por ejemplo, pues en vez de definir una "n" usaremos Y.

Entre las de tipo *PRIVATE* está "punt", es el resultado de calcular asesinatos-muertes en el registro en el que se está, para escribirlo con un *WRITE_INT*.

Antes de explicar las siguientes variables aclararé una cosa. Cuando usamos un *WRITE_INT* lo que le tenemos que asignar no es la variable (el nombre, digamos el identificador de la variable) sino la "posición de memoria" en que se encuentra ésta, fácilmente localizable con *OFFSET*. Si usamos un dato tipo *INT* no hay problema (solo con ver el nombre de la función *write_INT...*) pero claro, no podemos usar el *OFFSET* con una tabla de datos tipo WORD o tipo BYTE, porque como no ocupan una alineación entera de memoria (4 bytes) no se puede "localizar" la posición exacta dentro de la alineación, con *OFFSET*.

Entendido esto, comprenderemos para qué sirven *muerte1, muerte2, atrib1, atrib2, atrib3* y *atrib4*. Sirven para almacenar temporalmente los valores de *muerte[1]* y *atrib[3]* del registro que se está mirando, y así poder hallar el *OFFSET* de estas últimas variables.

Por último encontramos una variable llamada "ratón". Es muy simple, "ratón" vale 1 cuando se está pulsando el botón, y 0 cuando no. Al final del bucle, antes del *FRAME*, se comprueba si no se está pulsando, para ponerlo a 0. No podemos usar *IF* (mouse.left) porque lo que queremos saber es si tras haber pulsado una vez en algún botón se mantiene pulsado todavía el botón. Para que con sólo pulsar en el botón de "siguiente bot" un segundo, no detecte 100 pulsaciones y por lo tanto avance 100 bots (en la práctica es imposible, porque el juego no irá nunca a 100 fps justos. Aunque no hay 100 bots, daría la vuelta 3 veces), sino que hasta que no se suelte y se vuelva a pulsar el botón, no avance otro registro. Esto es aplicable a todos los botones.

Por último antes del bucle, indica que el fichero FPG a usar es MISC.FPG (ID=fpgmisc).

El bucle principal de la función *gestorbots()*

El bucle podemos descomponerlo así:

```
REPEAT

gráfico de mouse=predeterminado;

IF (x==-1)
writes
botones para el jugador;
ELSE
writes
botones para cualquier bot;
END

IF (se pulsa en botón izq. o se pulsa cursor izq.)
retrocede un bot;
END
IF (se pulsa en botón dcha. o se pulsa cursor dcha.)
retrocede un bot;
END
IF (se pulsa en botón inicio)
al inicio;
END

IF (no se pulsa ratón)
ratón=0;
END

FRAME;

borra_textos;
UNTIL (key(_esc));
```

Así parece muy simple, pero claro, al desarrollarlo queda una función un poquito amplia. Sin embargo, no deja de ser simple y fácil de entender. Quizá la parte más tediosa es el primer *IF...ELSE*, por su longitud, pero en cuanto lo veáis descubriréis que es bastante repetitivo y enseguida se asimila.

Para el ratón usamos *mouse.graph*, que es más fácil que un proceso dedicado al ratón. Como al pasar por encima de un botón (salvo los de adelante y atrás) el ratón adquiere otro gráfico, hay que definir al principio del bucle que el gráfico, hasta que se compruebe lo contrario, es el predeterminado (en nuestro caso el 990).

El primer bloque *IF...ELSE...END* tiene dos partes: la primera, el *IF (IF (x==-1))*, que actúa cuando se está viendo el registro del jugador, y la segunda, *ELSE*, en caso contrario, que sea cualquier bot.

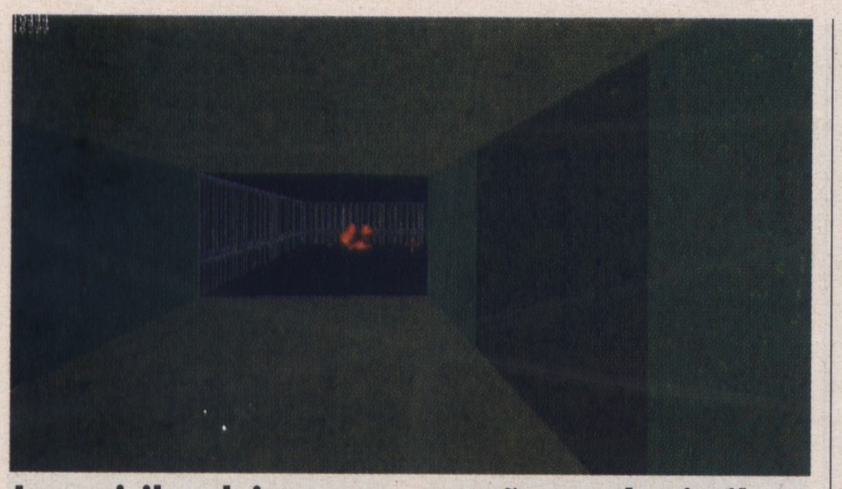
Comencemos con el primero. Para el jugador sólo necesitamos mostrar puntuaciones y nombre, puesto que lo demás no sirve. El que juega no puede aumentar su propia destreza con el ratón aumentando la "puntería" o su agresividad aumentando ésta misma, y de poco serviría elegir un *SKIN* para el jugador, ya que el juego transcurre en primera persona y no se ve al jugador desde fuera.

Lo primero es mostrar la "P" en el número de bot para indicar que el registro activo es el del jugador. Después escribe el nombre y las puntuaciones. Hay que tener en cuenta que estos textos se borran justo tras el *FRAME* para volver a reescribirlos en cada bucle, es decir, que se actualizan constantemente por si son modificados, e igual para con los bots.

En este caso sólo nos interesa saber si el ratón está sobre el botón de "cambiar nombre" (a la derecha del nombre, arriba en el centro) o si está en la cruz que indica "resetear" (resetear las puntuaciones). Esto viene en forma de dos bloques *IF* que comprueban si la X y la Y del ratón se encuentran dentro del cuadrado que forma cada botón. Si por ejemplo se está sobre el botón de cambiar nombre, el gráfico del ratón se cambia a este tipo de cursor ("cambiar"), y dentro de este *IF*, si se pulsa el botón izquierdo y RATÓN NO ES 1, se llama a la función *INPUT* para que retorne el

Una exclusiva captura aérea de la plaza (DEFAULT.WLD).

El búnker hará las delicias de los francotiradores.



Los misiles dejan una pequeña estela similar a la de Quake.

resultado en "nombre" (la variable que contiene el nombre del jugador). Para que lo veáis más claro:

```
IF (mouse.x>485 AND
mouse.x<502 AND mouse.y>11 AND
mouse.y<28);
    mouse.graph=992;
        IF (mouse.left AND ratón==0);
        nombre=input(319,20,4,15,
nombre,fuentemsj);
        ratón=1;
        END
    END
```

Sin comentarios, pero para el caso es lo mismo. 992 es el número de gráfico que contiene el cursor "cambiar", y los parámetros de la función *INPUT* son comentados más abajo. Para el botón resetear no hay más que poner a 0 los registros de la tabla *muerte[1]*.

Para el caso de los bots, es muy parecido, pero añadiendo más botones, esto es, los de *SKIN*, y todos los atributos, desde "agresividad" hasta "arma preferida". Tienen un par de flechas-botón cada uno, que sirven para aumentar y disminuir cada atributo. Salvo la de "arma preferida", que sólo puede oscilar entre 0 y 7, no nos preocupamos si ya estaba pulsado el ratón o no, porque avanzar de 0 a 100 teniendo que hacer 100 clics es realmente aburrido.

Con el caso "resetear puntuaciones" tenemos un problema. Los bots están siempre ordenados de mayor a menor puntuación neta. Si reseteamos la puntuación de uno a 0, tenemos que volver a ordenarlos, porque seguramente varios cambien de posición (el reseteado se irá al final y los que había en medio avanzarán un puesto). Por ello hay un pequeño bucle *FROM* que se encarga de calcular las "variables puntos" de cada registro de *bot[29]*, para ordenar esta estructura luego con QSORT.

Por último, con "cambiar skin" la ponemos en tipo de letra del sistema, para que ocupe menos y quepan rutas y nombres largos, y hay que tener en cuenta que hay que incluir ruta completa (a menos que esté en el directorio del ejecutable) y extensión del archivo. Eso permite utilizar *FPGs* con distinta extensión.

Fuera del primer bloque tenemos los dos *IFs* que se encargan de comprobar si se pulsa en los botones de "atrás" o "adelante", o se pulsa izquierda o derecha, para avanzar o retroceder un número de bot, desde -1 a 29, siendo cíclico (al pulsar adelante en el 29 salta al -1, y viceversa).

También tenemos que si se pulsa en el botón "inicio" salta automáticamente al registro -1, es decir, el del jugador.

Y ya para finalizar, la comprobación, que si no se está pulsando el botón del ratón "ratón" se vuelve 0.

Tras el bucle

Cuando se pulsa ESC, se acaba el bucle y hay que salir del gestor de bots. Obviamente, hay que salvar los cambios, si no de poco habrán servido. Para ello estas líneas:

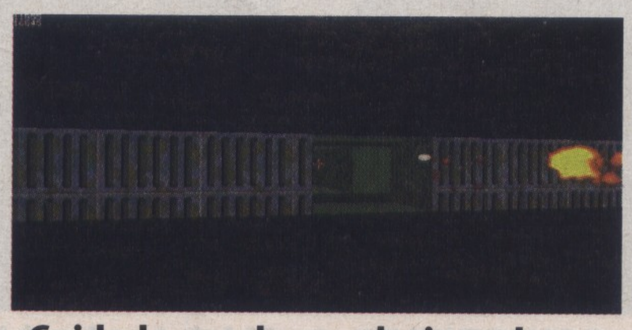
```
    fichero=fopen("bots.cfg","r+");
    fseek(fichero,0,0);
    fwrite(OFFSET
muerte,sizeof(muerte),fichero);
    fwrite(OFFSET
nombre,sizeof(nombre),fichero);
    fwrite(OFFSET
bot,sizeof(bot),fichero);
    fclose(fichero);
```

He vuelto a obviar los comentarios. Para más información, están en el PRG.

Y al final, se quita el gráfico del ratón con *mouse.graph=0* y se asigna 0 a estado, para decir que estamos de vuelta en el menú.

La función Input

La función *input*, como habéis podido comprobar, no viene comentada en el PRG. En realidad, está formada por dos funciones: *INPUT* (la principal) y *S_ASCII* (un apoyo que aumenta la velocidad). La misión de la función input es coger una entrada de caracteres del teclado y guardarla en una string, y una vez se pulsa intro, retornarla. Por ejemplo, si la llamamos con *pepe=input(taltaltal)*, al escribir algo y pulsar intro, "pepe" tomará esa cadena de texto (como una string).



¡Cuidado con las explosiones!

La función *input* no reconoce todos los caracteres; está adaptada a Div Deathmaker para reconocer sólo los necesarios.

La original función *input* es obra de *DEF*, y ésta es una versión *TAN* modificada que casi parece una función distinta. Pero aun así, está modificada a partir de la rutina de *DEF*. Ésta usaba la función *ascii()* de Div 2, pero gracias a la función de Tizo *S_ASCII* usa ahora ésta, reconociendo caracteres mucho más rápido.

Podéis considerar estas dos, al fin y al cabo, como una especie de "librería", así que no os tenéis que preocupar si no entendéis el código. Los parámetros son:

- X e Y: mientras se escribe, el texto se muestra en pantalla. Hay que decir dónde.
- CENTRADO: especifica el código de centrado del texto, como en los *write()*.
- EXTENSION: define la extensión máxima del texto a entrar.
- STRING PRCAD: a veces no se trata de coger un texto desde 0, sino de modificar uno ya existente. Si aquí introducimos "juanito", al escribir ya tendremos escrito "juanito".
- FUENTE: aquí es necesario especificar el *ID* de la fuente a usar. Si es 0, usará la fuente del sistema, la predeterminada de Div.

En el CD

En este número, en el directorio de la sección 3D/RED se encuentran todos los archivos ya disponibles en el anterior número; y además:

- DEFAULT.WLD: el mapa estándar de Div Deathmaker "Death Plaza".
- DEFAULT.TXT: el TXT correspondiente a este escenario DEFAULT.WLD.
- DEFAULT2.TXT: el TXT del SKIN base "DEFAULT.FPG".

Apaga y vamonos

Esto es todo por el momento. Sé que ya lleváis dos números (tres con este) ansiosos por ver algo jugable, pero tenéis que admitir que es mejor ir por partes. Conocéis lo de "vísteme despacio que tengo prisa", supongo... En el próximo número, SÍ, podréis moveros por el mapa elegido. Recordad que si tenéis alguna sugerencia / duda, lo que sea, me lo podéis mandar a mi e-mail, las 24 horas del día los 365 días del año. ¡Hasta el próximo número!

Ferminho (Fermín Vicente)
ferminho@lycosmail.com

Perfeccionando el diálogo

Conversaciones "de película"

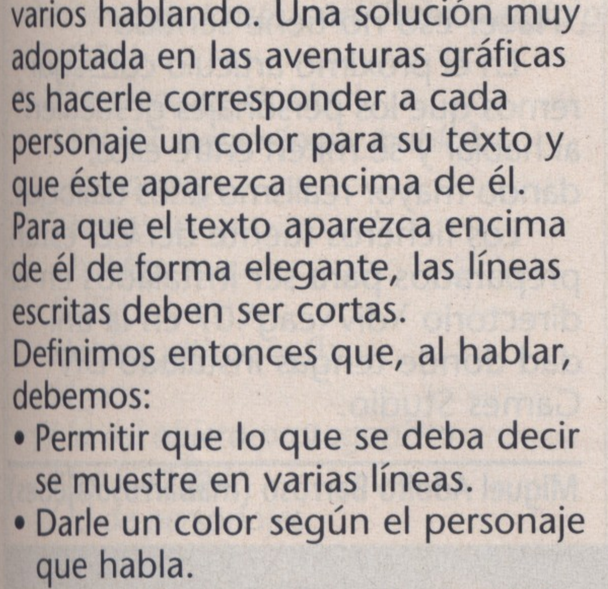
Ya estamos preparados para mejorar la forma de hablar los personajes de nuestra vídeo aventura. Hasta ahora hemos simulado el hablar con las funciones provisionales habla y habla2. Sin embargo, estas funciones provisionales quedan muy lejos de lo que en realidad deseamos

Esto nos ha servido para hacernos una idea de cómo podría quedar finalmente y permitirnos, mientras tanto, perfeccionar el menú de opciones, el tratamiento de acciones y los diálogos. Pero ya es hora de definir qué es "hablar".

Los personajes de nuestra aventura pueden estar en cualquier lugar del escenario, y puede haber varios hablando. Una solución muy adoptada en las aventuras gráficas es hacerle corresponder a cada personaje un color para su texto y que éste aparezca encima de él. Para que el texto aparezca encima de él de forma elegante, las líneas escritas deben ser cortas. Definimos entonces que, al hablar, debemos:

- Permitir que lo que se deba decir se muestre en varias líneas.
- Darle un color según el personaje que habla.
- Situar el texto sobre el personaje que habla.

Para dar mayor flexibilidad, podemos aprovechar para que, a diferencia de *dialoga*, *habla* permita al jugador realizar acciones mientras se ejecuta. Además, nos conviene que simule el diálogo entre personajes. Esto nos permitirá conseguir diferentes detalles muy llamativos en nuestra vídeo aventura, como, por ejemplo, que el protagonista escuche conversaciones de terceros o que pueda hacer diferentes acciones mientras uno o varios personajes hablan. Para conseguir este efecto, *hablar* debe ser totalmente independiente. Esta independencia se la damos al ejecutarlo como un proceso. El proceso entrará en un bucle *loop* mientras duren los textos, cediendo la ejecución con *frame*. Una vez finalizados los textos, el proceso finaliza y muere, volviendo a ser lanzado cuando vuelva a ser necesario. Debemos tener en cuenta que, a priori, no podemos determinar la cantidad de líneas de textos que debe representar *habla*. En el número anterior sólo preveíamos una línea. Por otro lado, mientras *habla* se está ejecutando, puede ocurrir que necesitemos añadirle más líneas de texto. Una buena solución es emplear una cola.



Colas

En el caso que nos preocupa, la función *habla* no sólo debe conocer los datos imprescindibles para representar el texto, sino que, además, precisa establecer un orden en ellos. Este establecimiento de orden nos lo otorga una estructura de tipo cola. Se debe pensar en una cola precisamente por su nombre. Si nos fijamos en cualquier cola (cola del metro, del autobús, del cine...), nos percatamos de que el primero en dejar la cola es quien llegó primero, y que todos los nuevos en la cola se sitúan al final para salir por el principio.

Se puede implementar una cola de muchas formas, nosotros emplearemos una lista de elementos y un puntero. El primer elemento será el de posición 0 en la lista. El puntero tendrá como valor la posición que ocupa el último elemento en la cola, valdrá cero si la cola está vacía.

Para extraer elementos de la cola lo que hacemos es recorrerla desde el principio hasta el final

Hay dos funciones para una cola, que podríamos llamar *acolar* y *desacolar*. La primera simplemente sitúa un nuevo elemento al final de la cola. La segunda devuelve el primer elemento de la cola y lo elimina de la lista. Nuevamente hay muchas formas de hacer esto, moviendo todos los elementos a una posición más avanzada (el segundo al primero, el tercero al segundo, etc.); usando un puntero para determinar al primer elemento, etc.

La cola que vamos a usar es un tanto peculiar. Tenemos la función *acolar* llamada *habla*. La función *desacolar* va incrustada en *habla_ejecuta* de una forma muy singular. Para extraer elementos de la cola lo que hacemos es recorrerla desde el principio hasta el final y, una vez acabada, la reiniciamos poniendo el puntero apuntando al principio. Esto

nos simplifica la programación, ya que, en nuestro caso, una vez que empezamos a sacar elementos no debemos parar hasta acabar con todos. También hay un inconveniente a tener en cuenta. Mientras estamos extrayendo elementos de la cola puede ocurrir que otros procesos sigan introduciendo elementos en ella. Puesto que la cola no se libera hasta haber llegado al último elemento debemos de cuidarnos de no intentar acolar más elementos de los posibles.

Es importante que los textos de varias líneas entren en la cola al revés

Planeando

Para este ejemplo vamos a emplear tres objetos personaje únicamente. El primero representará al protagonista, y en él no necesitamos programar nada. Los otros dos representan a un personaje cada uno, y son similares a otros que hemos tratado. Para comprobar que el texto realmente aparece sobre el personaje, los preparamos para moverlos con las teclas cursoras:

Protagonista: 1, 2, 3, 4
Personaje primero: Q, W, E, R
Personaje segundo: A, S, D, F

Justo al empezar el programa haremos que los personajes primero y segundo hablen entre ellos, entonces el protagonista los interrumpirá, dará la bienvenida y todo quedará en silencio. Durante este tiempo se podrá usar el menú de opciones. Sin embargo, claro está, si intentamos hacer algo, lo que el personaje nos vaya a decir irá a la cola.

Una vez empezado el juego, podremos hablar con un personaje u otro. Cuando hacemos la primera de las preguntas a cada uno de los personajes ésta cambiará por otra relacionada. En determinadas respuestas de un personaje, el otro puede entrometerse.

Implementando

Para los diálogos creamos tres nuevas fuentes, cada una de un color, para cada uno de los personajes. Establecemos el tamaño máximo de la cola con *max_habla* y definimos la estructura de la cola:

```
// Estructura Habla
STRUCT Hablar[max_habla]
  personaje;
  texto;
  habla;
END;
```

En la cola almacenaremos el *id* del proceso que habla (personaje), el texto que ha de decir (texto) y si debe ejecutarse el habla o no (habla). Cada entrada en la cola será una línea de texto al hablar. Todas las líneas con *habla=FALSE* se representarán a la vez que la primera *TRUE*. Es decir, si queremos que el personaje hable tres líneas que digan: "Hola", "bienvenido" y "a DIV"; las dos primeras valdrán *FALSE* y la última *TRUE*, llamando a la función como sigue:

```
habla(id_personaje, "a DIV", FALSE);
habla(id_ personaje, "bienvenido", FALSE);
habla(id_ personaje, "¨Hola,", TRUE);
```

Es muy importante, en los textos de varias líneas, que entren en la cola al revés, ya que a la hora de visualizarlo, la primera línea será la de más abajo. A medida que se acolan textos, si no se está hablando y se acola un texto TRUE, se lanza el proceso a ejecutar *habla_ejecuta*. La forma de funcionar de este proceso es muy simple:

1. Elemento es el primero de la cola (menos uno).
2. Coge siguiente elemento, si no hay más ve a 5 y haz la pausa.
3. Imprime sobre el personaje con su color, y encima de la anterior línea si la hubo.
4. Si no es *TRUE*, vuelve a 2.
5. Si es *TRUE* haz pausa de espera.
6. Borra el texto escrito.
7. Si hay más elementos vete a 2.

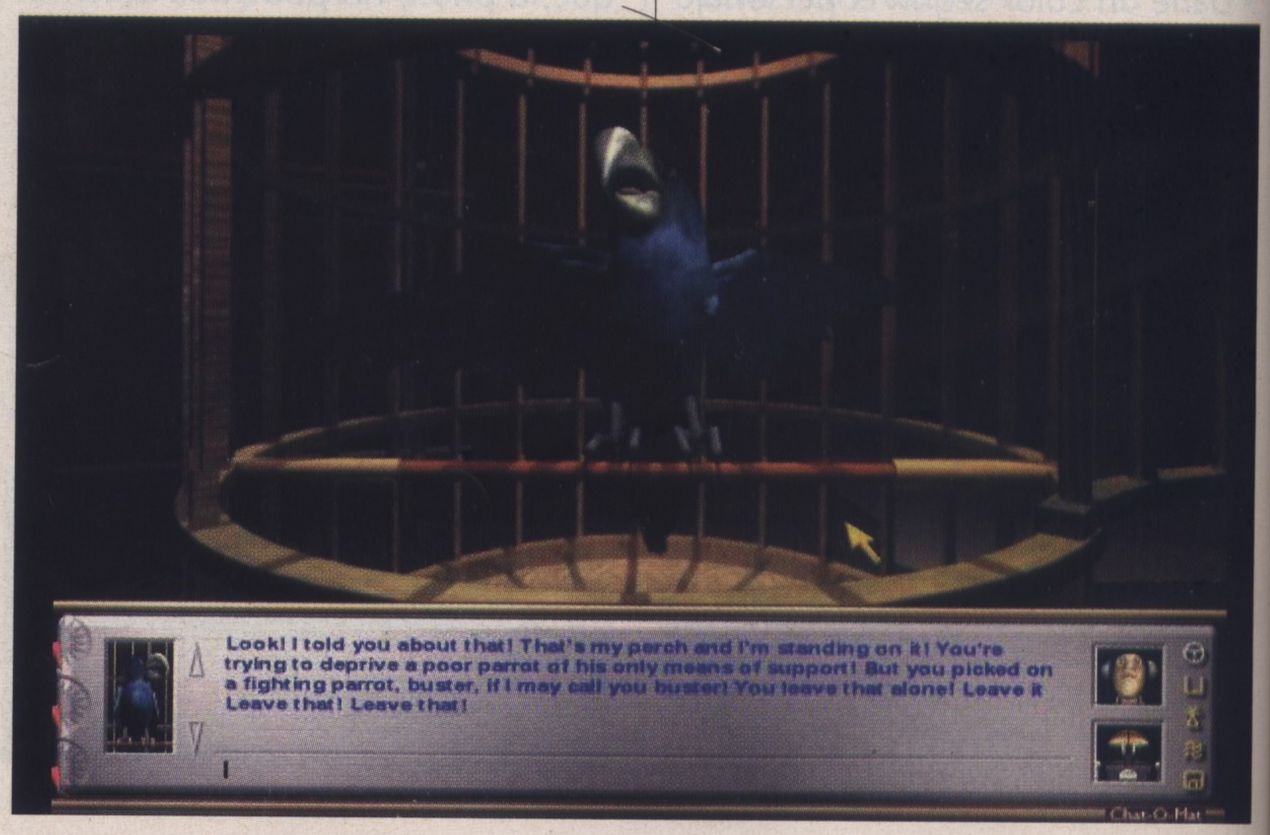
Hemos creado una fuente para cada personaje, todas iguales excepto en su color. El *id* de la fuente almacenado en el propio proceso del personaje (*letra*) y, de esta forma, en el momento de escribir, nos basta con usarlo recuperándolo de la forma *personaje.letra.* En DIV las coordenadas gráficas del proceso corresponden a su centro. La coordenada horizontal del texto coincide con la del personaje (*personaje.x,*). La coordenada vertical será ligeramente superior a la del personaje, debiendo sumarle la mitad del tamaño del personaje más la diferencia que queramos. Además, debemos saltarnos tantas líneas como líneas hayamos escrito ya, haciendo: *personaje.y-.personaje.height/2-10-10*(linea+1)*, o sea, coordenada vertical – mitad del alto del personaje menos 10 menos 10 tantas veces como línea sea. Debemos restar para que salga arriba, sumar si queremos que salga debajo del personaje.

Para facilitarnos la programación crearemos una función *hablap* a la que sólo le pasaremos el texto. Esta función nos servirá para textos que deba decir el protagonista y que sean de una sola línea, como por ejemplo "No puedo llevarlo", "Hacer eso no tiene sentido".

En el próximo artículo conseguiremos que los personajes gesticulen al hablar y se miren entre ellos, dando mayor realismo a los diálogos.

Los ficheros fuente del CD están preparados para ser instalados en el directorio \div\cag\07 en la unidad donde tengas instalado DIV Games Studio.

Miguel Adolfo Barroso (mabarroso@jet.es)



Nota sobre los ficheros fuente

```
// Inicia Hablar
 habla_inicia();
 id_pro = protagonista(150, 250, f_objetos);
 id_p1  = per1 (300, 250, f_objetos);
 id_p2  = per2 (450, 250, f_objetos);
 habla(id_p1,  "Hola Verde", TRUE);
 habla(id_p2,  "Hola Marron", TRUE);
 habla(id_p1,  "que estás por aquí?", FALSE);
 habla(id_p1,  "¿Es la primera vez", TRUE);
 habla(id_p2,  "Si, ¿y tú?", TRUE);
 habla(id_pro, "PSSSTTT...", TRUE);
 habla(id_p1,  "¿eh?", TRUE);
 habla(id_p2,  "¿oh?", TRUE);
 habla(id_pro, "silencio por favor", FALSE);
 habla(id_pro, "Un momento de ", TRUE);
 habla(id_p1,  "De acuerdo", TRUE);
 habla(id_p2,  "Vale", TRUE);
 habla(id_pro, "amigo...", FALSE);
 habla(id_pro, "Hola", TRUE);
 habla(id_pro, "más los diálogos", FALSE);
 habla(id_pro, "ahora te gusten", FALSE);
 habla(id_pro, "Espero que", TRUE);
```

Código antes del bucle principal para simular el diálogo automático entre personajes.

```
CONST
 max_habla      = 24;          // Máximo de frases para hablar en la pila
GLOBAL
 // Estructura Habla
 STRUCT Hablar[max_habla]
  personaje;
  texto;
  habla;
 END;
 Hablar_ultima = 0;
 id_pro;
// Id del objeto protagonista
 id_p1, id_p2;
// Id de personajes
LOCAL
 letra          = 0;

PROCESS habla_inicia();
 BEGIN
  Hablar_ultima = 0;
 END;

PROCESS habla(per, tex, hab);
 BEGIN
  Hablar[Hablar_ultima].personaje    =    per;
  Hablar[Hablar_ultima].texto        =    tex;
  Hablar[Hablar_ultima].habla        =    hab;
  Hablar_ultima++;
  IF ((hablando==FALSE) && (hab==TRUE)) habla_ejecuta(); END;
 END;

PROCESS hablap(texto);
 BEGIN
  habla(id_pro, texto, TRUE);
 END;

PROCESS habla_ejecuta();
 PRIVATE
  t, i, j, jj;
  tid[max_habla];
 BEGIN
  hablando=TRUE;
  j=0; jj=0;
  LOOP
   IF (j>=Hablar_ultima) break; END;
   jj=0; t=0;
   LOOP
    IF (j>=Hablar_ultima) break; END;
    tid[jj++]= write (Hablar[j].personaje.letra,
                      Hablar[j].personaje.x,
                      Hablar[j].personaje.y-Hablar[j].personaje.height/2-10-10*(jj+1),
                      4,
                      Hablar[j].texto);
    t = t + ascii_len(Hablar[j].texto);
    IF (Hablar[j].habla==TRUE) Break; END;
    j++;
   END;
   FOR (i=0; i<t*10; i++)
    FRAME(r_habla);
   END;
   WHILE (jj>0)
    delete_text(tid[—jj]);
   END;
   j++;
  END
  hablando=FALSE;
  Hablar_ultima = 0;
 END;
Código para las funciones habla

CASE o_Hablar: frase=-2;
     LOOP
      WHILE (hablando)
      FRAME;
      END;
      SWITCH (frase)
      CASE -2:
       IF (d_nombre == 0)
       dialoga("¨Te llamas Verde?",
                1,
         "Es un nombre muy tonto",
                2,
         "El verde no es mi color favorito",
                3,
         "Adios",                    0);
       ELSE
       dialoga("¨Ver y De?",
                4,
         "Es un nombre muy tonto",
                2,
         "El marron no es mi color favorito", 3,
         "Adios",                    0);
       END;
      END;
      CASE -1:
       FRAME;
      END;
      DEFAULT:
       SWITCH (dialogo[frase].valor)
       CASE 0:
        habla(ID, "Hasta pronto", TRUE);
        break;
       END;
       CASE 1:
        habla(ID, "Si...", TRUE);
        habla(ID, "llamaba Ver", FALSE);
        habla(ID, "Mi padre se", TRUE);
        habla(ID, "llamaba De", FALSE);
        habla(ID, "Mi madre se", TRUE);
        habla(ID, "de nombre Verde", FALSE);
        habla(ID, "Asi que me pusieron", TRUE);
        d_nombre = 1;
       END;
       CASE 2:
        habla(ID, "como tu", FALSE);
        habla(ID, "tener nombre", FALSE);
        habla(ID, "M s tonto es no", TRUE);
        habla(id_p1, "¿Lo teñimos?", TRUE);
        habla(ID, "da pena", FALSE);
        habla(ID, "en el fondo me", FALSE);
        habla(ID, "Dejalo, si", TRUE);
       END;
       CASE 3:
        habla(ID, "Tu te lo pierdes.", TRUE);
        habla(ID, "ninguna esperanza", FALSE);
        habla(ID, "Tus gustos no tienen,", TRUE);
       END;
       CASE 4:
        habla(ID, "Claro...", TRUE);
        habla(ID, "Asi puedes...", TRUE);
        habla(ID, "VERDE", FALSE);
        habla(ID, "VER me DE color", TRUE);
        habla(id_p1, "Ja, Ja, Ja", TRUE);
        habla(ID, "Je, Je, Je", TRUE);
        d_nombre = 0;
       END;
        END
        frase=-2;
      END;
     END;
     activa_menu();
    END;
```

Fragmento de control de diálogos de un personaje

Juegos de Estrategia (VII)

Retomamos nuestro Parser

En el número cinco de esta revista habíamos empezado la programación de un pequeño parser para poder editar los mapas de pruebas de una forma más sencilla. En este número retomamos el parser para dotarlo de más funcionalidad.

Habíamos dejado nuestro parser de tal manera que se podía crear un mapa y añadirle mobs de distintas clases. También podíamos grabarlo y recuperarlo en un fichero fijo. Teníamos el problema que ni se podían borrar los mobs que hubiésemos puesto, ni podíamos decidir a que bando pertenecían. Pues ya lo tenemos solucionado.

Seleccionando los bandos

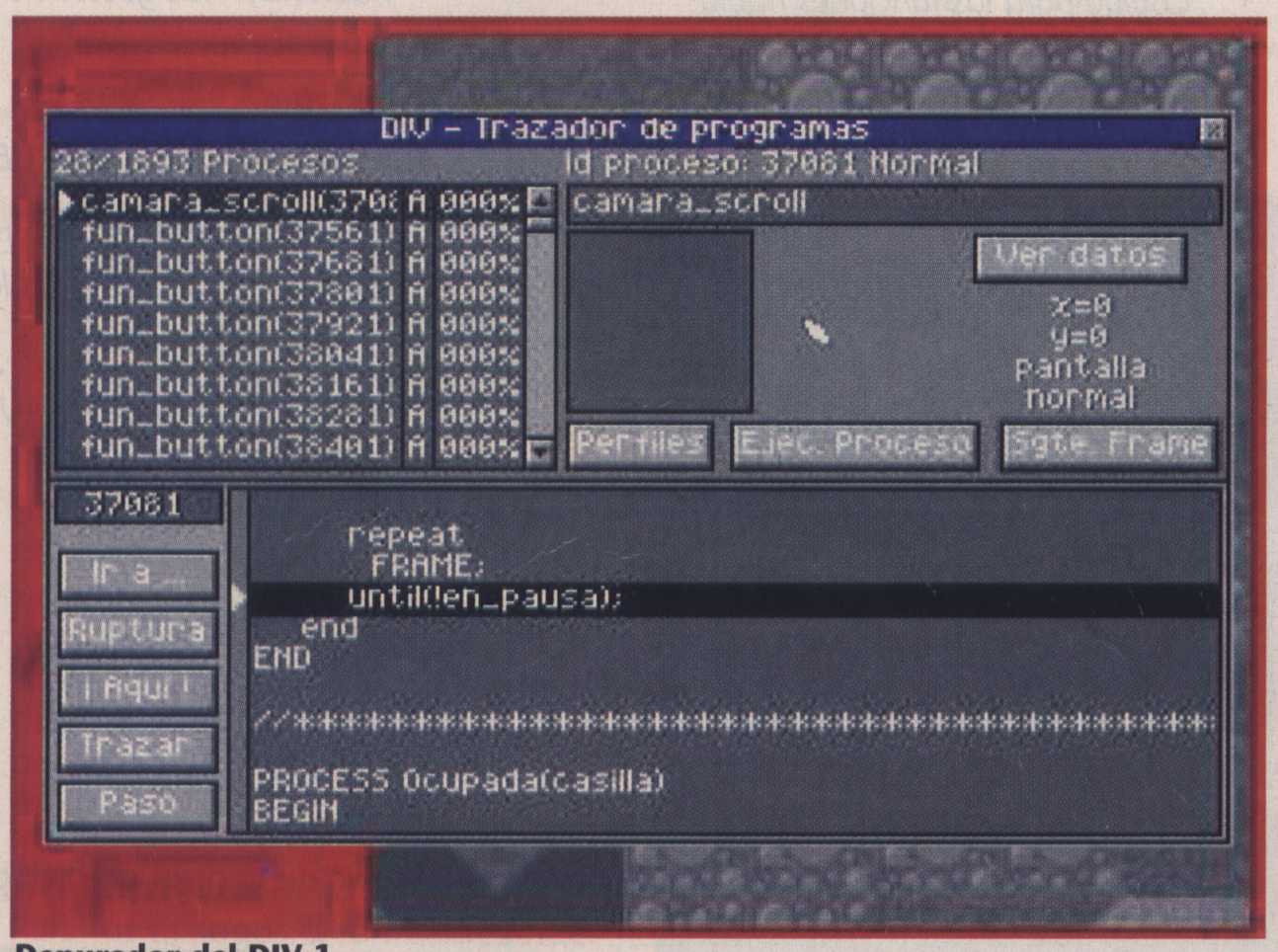
Hemos decidido, por sencillez, permitir solo dos posibles bandos. En un alarde de imaginación sin limites se ha llamado a los bandos "los buenos" y "los malos", claramente diferenciables entre sí por el color verde de *los buenos* frente al morado de *los malos*. Para diferenciar a los mobs en el mapa, hemos codificado un proceso que seguirá a su padre desde su nacimiento hasta la tumba. Este proceso muestra un recuadro de color morado o verde, según el bando, de un tamaño variable según la vida relativa que le quede a nuestro mob. El cálculo del tamaño del recuadro es tan sencillo como asignarle a la variable *size* del proceso el porcentaje de vida que le queda al mob.

Este porcentaje se calcula por una regla de tres muy simple: *vida_actual*100 / max_vida*. Se limita el tamaño mínimo del cuadro para que no desaparezca antes de que el mob haya muerto, ya que, si desaparece, no sabríamos a que bando pertenece. El proceso se ejecuta sobre un bucle infinito, por lo que es el padre quien debe ocuparse de matarlo cuando sea necesario (muerte del padre). Aprovechando la ocasión, comentaremos que siempre se debe evitar la existencia de procesos huérfanos, ya que si por alguna razón intentan acceder a una variable local del padre, el programa entero cascaría. Por esta razón, debemos evitar utilizar la señal *S_KILL* y utilizar siempre *S_KILL_TREE*, con lo que evitamos la existencia de procesos huérfanos.

Siguiendo con el tema que nos ocupa, el proceso que pinta la vida y el bando es hijo del mob del cual pinta su vida y bando, por lo que es éste, en su fase de inicialización (antes del bucle), el que lo llama.

Para seleccionar a que bando pertenece un mob, hemos añadido dos botones a la interfaz del parser, "buenos" y "malos", y que funcionan igual que los de selección entre mobs y suelos. Al pulsar uno, todos los mobs que pongamos a partir de ese momento pertenecerán a ese bando. Inicialmente el bando por defecto es el de "los buenos", así que, cada vez que entremos en el parser y queramos poner un mob "malo", deberemos pulsar el botón de "malos".

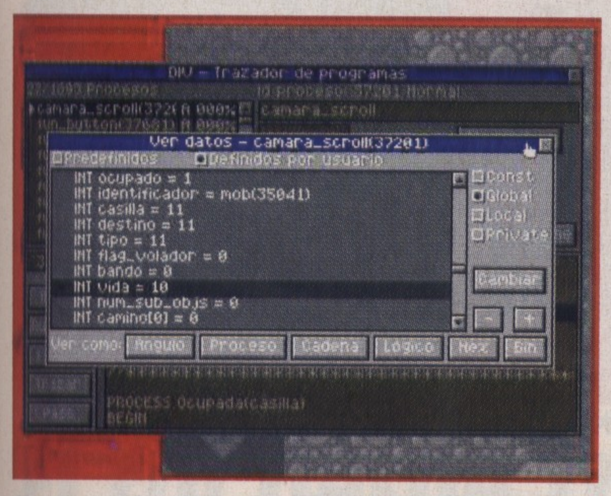
Y una vez solucionado el problema de los bandos, queda el de borrar los mobs existentes, así que vamos allá.



Depurador del DIV 1.

Borrado de mobs existentes

Hasta ahora, los mobs que poníamos en el mapa y salvábamos se quedaban ahí, lo cual no era muy correcto. Por esto hemos añadido la posibilidad de borrar los mobs existentes. Desde el punto de vista del usuario, es algo tan sencillo como pinchar con el botón derecho del ratón sobre un mob existente. Es preciso que estemos en modo inserción de mobs, por lo que no podremos borrar ningún mob mientras estemos poniendo suelos. Esto sirve como sistema de seguridad para evitar el borrado accidental de un mob mientras editamos el mapa del suelo.



Ventana de "Ver Datos" en el depurador.

Como siempre, lo que es sencillo para el usuario, no lo es tanto para el desarrollador, así que veamos cómo hacerlo. Se podría poner un control en el proceso mob de tal manera que, cuando estemos en modo parser y haya colisión con el ratón y además se pulse el botón derecho, entonces nos suicidemos; pero esta solución, aunque fácil, carga al proceso mob con una responsabilidad que no es suya y que lo ralentiza con una comprobación inútil cuando estamos jugando. Lo más correcto sería hacerlo desde el bucle del parser, pero entonces la detección de la colisión entre el mob y el ratón no es detectable. Esta es la razón de que tengamos que comprobar sobre qué casilla del mapa se ha pulsado, y a partir de esta información, averiguar que mob está en ella.

El problema está en que, con la información que tenemos guardada hasta el momento, tendríamos que recorrer la tabla de mobs comprobando uno a uno si ese mob está en esa casilla, para después matarlo. Esto, aunque funciona, ralentiza el proceso, así que deberemos gastar un poco más de memoria para guardar una matriz con los *num_id* (índices de la tabla de mobs) de los mobs que haya sobre cada casilla. Imaginad que es como otra capa del mapa de juego. Con esta información, en un solo acceso sabremos que mob debemos borrar. Por supuesto, cualquiera de las otras dos soluciones es válida. Hemos optado por la última solución, porque la política elegida es que prime la velocidad sobre el gastos de memoria. Si lo que se desea es optimizar el gasto de memoria, se puede sacrificar la velocidad. La elección de esta política se debe a que el mayor gasto de memoria se produce por la carga de gráficos, siendo las variables lo que menor porcentaje de memoria utilizan. Por ejemplo, un gráfico pequeño de 20x20 (400 pixeles) ocupa lo mismo que la matriz en la que guardamos los *num_id* (10x10x4bytes = 400). Si lo que se desea es reducir el gasto de memoria, lo más recomendable es reducir el tamaño de los gráficos o el número de estos que utilizamos.

En un alarde de imaginación sin limites hemos llamado a los bandos "los buenos" y "los malos"

Ahora que ya podemos borrar mobs y que tenemos un parser bastante decente, vamos a probar la búsqueda de caminos que comentábamos en el número 4 y que no podíamos probar debido a la inexistencia de mapas.

Función que muestra el bando en la esquina superior derecha de un mob

```
PROCESS vida_y_bando(max_vida)
BEGIN   file=f_mobs;   loop
ctype=father.ctype;
    graph =
mobs[father.num_id].bando + 400;
    x=father.x + 7;
    y=father.y - 10;
    z=father.z - 100;
    size=
mobs[father.num_id].vida * 100 /
max_vida;
    if(size<20) size =20; end
    FRAME(100);
  end
END
```

Código relativo a la búsqueda de caminos en el proceso mob.

```
rejilla_con_mobs[mobs[num_id].casilla] = -1; /*libero mi casilla*/
  for(i=0;i<=10*10;i++)
     rejilla_busqueda[i]=-1;
  end

  //si debo ir a algun sitio y estoy parado...
  if(mobs[num_id].destino!=mobs[num_id].casilla&&!moving)

     if(no_se_como_llegar)

if(busca_camino(mobs[num_id].casilla,mobs[num_id].destino,num_id,0)==OK)
              no_se_como_llegar=false;
              mobs[num_id].next_sub_obj=1;

mobs[num_id].casilla=mobs[num_id].camino[mobs[num_id].next_sub_obj];
              moving=true;
           end
        end
     else
        if(moving)
           xy_reales(&x_dest,&y_dest,mobs[num_id].casilla);
           angle=fget_angle(x1,y1,x_dest,y_dest);
           if(fget_dist(x1,y1,x_dest,y_dest) > max_speed)
              x1+=get_distx(angle,max_speed);
              y1+=get_disty(angle,max_speed);
           else
              x1=x_dest;
              y1=y_dest;
              mobs[num_id].next_sub_obj++;
              if(mobs[num_id].next_sub_obj > mobs[num_id].num_sub_objs)
                 //ya no se como seguir
                 no_se_como_llegar=true;
                 moving=false;
              else

mobs[num_id].casilla=mobs[num_id].camino[mobs[num_id].next_sub_obj];
              end
           end

        else
           no_se_como_llegar=true; /* Estoy en destino */
        end
     end

     rejilla_con_mobs[mobs[num_id].casilla] = num_id;
     for(i=0;i<=10*10;i++)
        rejilla_busqueda[i]=-1;
     end
```

Probando la búsqueda de caminos

La búsqueda de caminos ha implicado algunos cambios en el código del proceso mob, así que se recomienda ver el cuadro donde aparecen estos cambios.

Si queremos ver como los mobs van de un sitio a otro, debemos ejecutar el programa desde el entorno DIV (1 ó 2) para poder "depurarlo". Lo que debemos hacer es entrar en el parser, para a continuación colocar un mob en algún lugar de un mapa previamente dibujado. Se recomienda que el mapa tenga montañas (burbujas) colocadas de la forma que más enrevesada os parezca y que el mob no sea volador (no pongáis una mosca porque vuela por encima de las burbujas). Una vez colocado el mob, pulsamos "F12" para que salte el debuger (depurador). Si alguien no lo ha utilizado nunca, este es el momento de aprender alguna cosilla.

Una vez en el depurador, tenemos varios botones y ventanas. Pulsamos sobre "Ver Datos", el cual sirve para ver los datos del proceso activo, las constantes, y las globales. Como lo que queremos cambiar es un dato global, desmarcamos las casillas "locales" y "privadas", y marcamos "globales". Tras esto, se deberían ver en la ventana las variables globales de nuestro programa con sus respectivos valores. Buscamos la estructura *mobs*, y dentro de ella el dato destino. Si tenemos varios mobs, debemos elegir el que queremos cambiar situándonos sobre *mobs[0]* y dando a los botones + y – hasta que aparezca el número del mob que queremos mover. Una vez hecho esto, nos colocamos sobre destino y pulsamos el botón "cambiar" lo cual abrirá una ventana en la que meteremos el nuevo valor de la variable. Pondremos el valor de la casilla a la cual queremos que se dirija el mob. Tras esto, salimos de todas las ventanas de depuración pulsando repetidamente sobre la cruz que hay en la esquina superior derecha. Veremos como el mob comienza a moverse hacia su destino.

Nueva interfaz del parser.

En el próximo número mejoramos la búsqueda para que no dar tantas vueltas, y empezaremos la parte más interesante: la IA.

Emilio Llamas Alba (YuMoK)

Función de búsqueda de caminos.

```
PROCESS
busca_camino(casilla_actual,casilla_de
stino,num_id,profundidad)
PRIVATE
    i;
    angulo;
    x1,y1,x2,y2;
    camino_encontrado=0;
    casillas[8]; /* casillas de alrededor
de un mob */
BEGIN
    /* Incrementamos la profundidad */
    profundidad++;
    /* Si destino alcanzado, marcarlo y
OK */
    if(casilla_actual==casilla_destino)
        if((profundidad <= SEARCH_BUF-
FER))

mobs[num_id].num_sub_objs=profun-
didad;
            mobs[num_id].camino[profun-
didad]=casilla_actual;
        else

mobs[num_id].num_sub_objs=SEARCH
_BUFFER;
        end
        return (OK);
    end
    /* Si recurrimos demasiado: ERROR
*/
    if (profundidad > MAX_PROFUNDI-
DAD_RECURSION)
        return (ERROR);
    end
    /* Si ocupada: ERROR */
    if(Ocupada(casilla_actual))
        return (ERROR);
    end

    /* Ocupar casilla */  rejilla_busque-
da[casilla_actual]=num_id;

    /* Calcular angulo */
    xy_reales(&x1,&y1,casilla_actual);
    xy_reales(&x2,&y2,casilla_destino);

    angulo = fget_angle(x1,y1,x2,y2);
    if (angulo<0) angulo+=360000;
end
  angulo/=45000;

    casillas[0]=angulo;
    casillas[1]=angulo+1;
    casillas[3]=angulo+2;
    casillas[5]=angulo+3;
    casillas[7]=angulo+4;
    casillas[2]=angulo-1;
    casillas[4]=angulo-2;
    casillas[6]=angulo-3;
    for(i=0;i<=7;i++)
        if (casillas[i]<0) casillas[i] += 8;
end
        if (casillas[i]>7) casillas[i] -= 8;
end

xy_matriz(&x1,&y1,casilla_actual);
        switch (casillas[i])
            case 0:
                x1++;
            end
            case 1:
                x1++;
                y1—;
            end
            case 2:
                y1—;
            end
            case 3:
                y1—;
                x1—;
            end
            case 4:
                x1—;
            end
            case 5:
                x1—;
                y1++;
            end
            case 6:
                y1++;
            end
            case 7:
                y1++;
                x1++;
            end
        end
        casillas[i]=posicion_xy(x1,y1);
    end//for
    /* Llamar a las casillas por orden de
cercania */
    for(i=0;i<=7;i++)
      if(casillas[i] != -1)        //casilla en
el borde
        camino_encontrado =
busca_camino(casillas[i],casilla_desti-
no,num_id,profundidad);

/* Si OK y multiplo de SEARCH_JUMP,
guardar en

mobs[].camino[profundidad/SEARCH_J
UMP] la casilla */
        if (camino_encontrado==OK)
            if((profundidad <=
SEARCH_BUFFER))
                mobs[num_id].camino[pro-
fundidad]=casilla_actual;
            end
            return (OK);
        end
      end
    end
    return (ERROR);
END
```

Vistas y Perspectivas en los RPG

El rol desde tres puntos de vista

Los juegos de rol son muy variados, y cada uno tiene sus propias características que lo hace diferente de los otros. Una de estas características, que nos permitiría clasificar a los RPG, sería la forma en la que nuestros personajes interactúan con el mundo en el que se mueven y el tipo de vista o perspectiva que posee el juego.

Este mes analizaremos los tres tipos de vista bidimensionales más simples.

Vista cenital 1

Este tipo de vista es la más sencilla y, de hecho, hoy ya no se utiliza por razones tan evidentes como la poca vistosidad y realismo que aporta al juego. El motivo de su gran uso hace unos años es bastante simple, se usaba porque resultaba muy sencilla de implementar y no requería de un ordenador muy potente.

La detección de colisiones se suele llevar acabo mediante un array bidimensional, o una matriz *sparse* (ver cuadro 1) si se quiere optimizar, formada por ceros y unos. Los ceros indican que el terreno esta libre y los unos indican un obstáculo.

Para generar el terreno se suelen usar tiles y el desplazamiento se realiza únicamente de casilla en casilla y sin ningún tipo de animación especial. Para implementar los tiles usaremos el mismo array de colisiones, de manera que no será necesario usar 2 arrays.

Un pequeño ejemplo de esto sería:

```
Program divmania_7_1;

Global

byte escenario[9,9]=
1,1,1,1,1,1,1,1,1,1,
1,0,0,0,0,0,0,0,0,1,
1,0,0,0,1,1,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,
1,0,0,1,0,0,1,0,0,1,
1,0,0,1,0,0,1,0,0,1,
1,0,0,0,1,1,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,1,
1,1,1,1,1,1,1,1,1,1;

Begin
tilea();
jugador();
End

Process jugador()

Private
int i,j,c;

Begin
graph=load_map("perso.map");
x=32+32/2;
y=24+24/2;
i=1;
j=1;
Loop
if(c==0)

if(key(_right) && !escenario[i+1,j])
x+=32;
i++;
c=5;
end

if(key(_left) && !escenario[i-1,j])
x-=32;
i--;
c=5;
end

if(key(_up) && !escenario[i,j-1])
y-=24;
j--;
c=5;
end

if(key(_down) && !escenario[i,j+1])
y+=24;
j++;
c=5;
end

else
c--;

end

frame;
End

End

Function tilea()

Private
t1,t2,t;
int i,j,x_tile,y_tile;

Begin
x_tile=32;
y_tile=24;
set_mode(m320x240);
t1=load_map("t1.map");
t2=load_map("t2.map");
t=new_map(320,240,320/2,240/2,0);
```


```
for(i=0;i<10;i++)
for(j=0;j<10;j++)
if(escenario[i,j]==0)
map_put(0,t,t1,i*x_tile+x_tile/2,j*y_til
e+y_tile/2);
else
map_put(0,t,t2,i*x_tile+x_tile/2,j*y_til
e+y_tile/2);
end
end
end
put_screen(0,t);
unload_map(t);
unload_map(t1);
unload_map(t2);
End
```

Este ejemplo sería uno de los esquemas más simples para un juego de rol. Está formado por la función *tilea* (ver el cuadro 2 para más detalles) y por el proceso *jugador*, cuyo único cometido es desplazar al personaje por la pantalla comprobando que no haya obstáculos.

Vista cenital 2

Este segundo tipo sería muy similar al anterior, pero contiene algunas modificaciones que lo hacen mucho más atractivo, como pueden ser el scroll y las animaciones del personaje.

Las animaciones del personaje serían muy simples, sencillamente sirven para dar más sensación de movimiento al juego. Otra cosa que se podría añadir es objetos que estuviesen en un plano de profundidad distinto al del jugador, como podrían ser el techo, nubes, barrancos, etc.

```
Program divmania_7_2;

Global

byte escenario[19,19]=
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,1,
1,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,0,0,0,1,
1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,0,0,1,
1,0,0,0,0,0,0,0,0,0,0,1,0,1,1,1,0,0,0,1,
1,0,0,0,0,0,1,0,0,0,0,1,0,1,1,1,0,0,0,1,
1,0,0,0,0,0,1,0,0,0,0,1,0,1,1,1,0,0,0,1,
1,0,0,0,0,0,1,0,0,0,0,1,0,1,1,1,0,0,0,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1;

Begin
tilea();
jugador();
End

Process jugador()

Private
int i,j,k;

Begin
ctype=c_scroll;
scroll.camera=id;
cnumber=c_0;
load_fpg("per.fpg");
graph=3;
x=32+32/2;
y=24+24/2;
i=1;
j=1;
Loop
if(key(_right) && !escenario[i+1,j])

i++;
angle=0;
for(k=0;k<3;k++)
x+=32/3+1;
graph--;
if(graph==0)
graph=3;
x--;
end
frame(300);
end

end

if(key(_left) && !escenario[i-1,j])
i--;
angle=180000;
for(k=0;k<3;k++)
x-=32/3+1;
graph--;
if(graph==0)
graph=3;
x++;
end
frame(300);
end
end

if(key(_up) && !escenario[i,j-1])
j--;
angle=90000;
for(k=0;k<3;k++)
y-=24/3;
graph--;
if(graph==0)graph=3;end
frame(300);
end

end
```


```
if(key(_down) && !escenario[i,j+1])
j++;
angle=270000;
for(k=0;k<3;k++)
y+=24/3;
graph--;
if(graph==0)graph=3;end
frame(300);
end

end

frame;
End

End

Function tilea()

Private
t1,t2,t3,ta,tb;
int i,j,x_tile,y_tile;

Begin
x_tile=32;
y_tile=24;
set_mode(m320x240);
t1=load_map("t1.map");
t2=load_map("t2.map");
t3=load_map("nube.map");
ta=new_map(640,480,320,240,0);
tb=new_map(640,480,320,240,0);

for(i=0;i<20;i++)
for(j=0;j<20;j++)
if(escenario[i,j]==0)
map_put(0,ta,t1,i*x_tile+x_tile/2,j*y_t
ile+y_tile/2);
else
map_put(0,ta,t2,i*x_tile+x_tile/2,j*y_t
ile+y_tile/2);
end
end
end

for(i=0;i<10;i++)
map_put(0,tb,t3,rand(0,640),rand(0,
480));
end

start_scroll(0,0,ta,0,0,0);
scroll.z=+1000;
nubes(tb);
unload_map(ta);
unload_map(t1);
unload_map(t2);
unload_map(t3);

End
```

```
Process nubes(t)

Begin
graph=t;
ctype=c_scroll;
x=320;
y=240;

Loop
frame;
End
End
```

Este ejemplo no es muy distinto del anterior, salvo que se le aplica una animación al personaje al moverse y se añade un proceso que pone unas nubes por encima del jugador.

Vista lateral

Esta tipo de vista es también muy sencilla de implementar. Consiste en un scroll lateral por el que los personajes se van moviendo lateralmente mediante unas animaciones medianamente elaboradas.

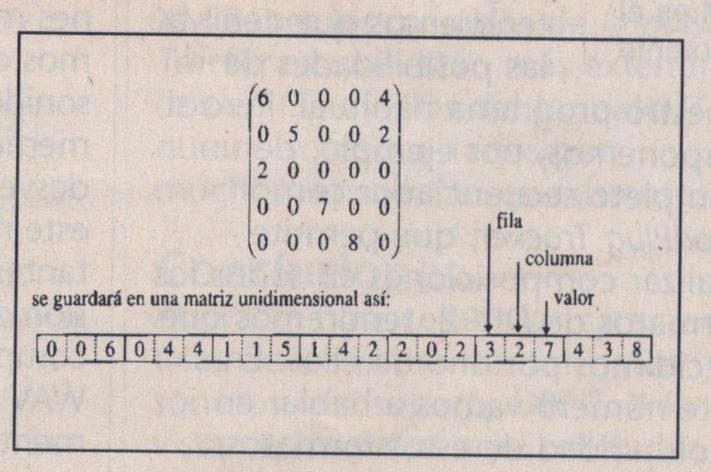
La detección de colisiones con el escenario se puede hacer mediante un mapa de durezas y la de los personajes mediante la función *collison*. En este tipo de vista también se pueden añadir objetos en planos *z* diferentes para dar más sensación al juego.

```
Program divmania_7_3;

Begin
set_mode(m320x240);
start_scroll(0,0,load_map("decor.map"
),0,0,0);
perso();
obj();
End

Process perso();

Private
f1,dur;
int mov;

Begin

f1=load_fpg("per1.fpg");
dur=load_map("decord.map");
graph=1;
scroll.camera=id;
x=10;
y=200;
ctype=c_scroll;

loop

if(key(_right))
if(map_get_pixel(0,dur,x+5,y)!=53)
x+=5;
end
mov=1;
graph++;
if(graph>5)
graph=1;
end
flags=1;
end

if(key(_left))
if(map_get_pixel(0,dur,x-5,y)!=53)
x-=5;
end
mov=1;
graph++;
if(graph>5)
graph=1;
end
flags=1;
end

if(key(_up))
if(map_get_pixel(0,dur,x,y-5)!=53)
y-=5;
end
mov=1;
graph++;
if(graph>5)
graph=1;
end
end

if(key(_down))
if(map_get_pixel(0,dur,x,y+5)!=53)
y+=5;
end
mov=1;
graph++;
if(graph>5)
graph=1;
end
end

if(mov==1)
mov=0;
else
graph=1;
end

frame;
end

unload_map(dur);
unload_fpg(f1);
End
Process obj()

Private
far;

Begin
far=load_map("far.map");
graph=far;
ctype=c_scroll;
y=150;
x=100;
clone
x=400;
end
clone
x=800;
end

loop
frame;
end

End
```

Ramón de España
maqnaziller@pagina.de

¿Qué es una matriz sparse?

Una matriz sparse es una matriz unidimensional en la que sólo se almacenan los elementos no nulos precedidos por sus índices. El uso de estas matrices reduce considerablemente el uso de memoria siempre y cuando haya una gran cantidad de elementos nulos.



Función Tilea

El objetivo de este artículo no es aprender a tilear, así que solo comentaré brevemente el proceso usado para el tileo. Pasos a seguir:

- Cargar los tiles mediante la función *load_map* (según el formato gráfico que usemos).
- Crear un array bidimensional en el que marcaremos qué mapa corresponderá a cada casilla.
- Crearemos una mapa con las dimensiones de nuestro terreno con la orden *new_map*.
- Mediante dos bucles *for* pondremos los tiles, usando la orden *map_put*, en el mapa que hemos creado posteriormente.
- Pondremos el mapa como fondo de pantalla con *put_screen*.

Descargaremos de la memoria los tiles y el mapa del escenario con *unload_map*.

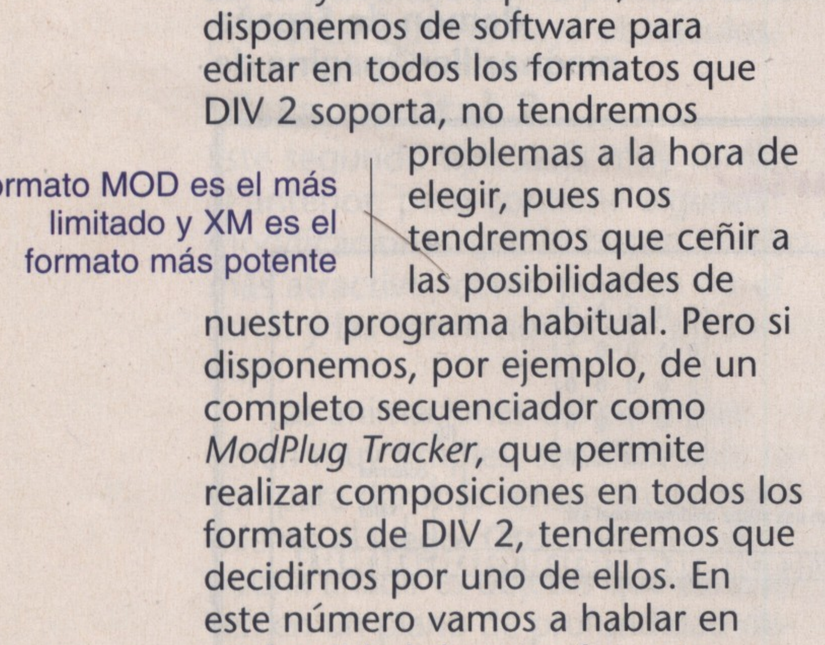
WAV, S3M, XM, MOD,...

Formatos musicales que soporta DIV 2

Tiempo atrás, cuando programábamos nuestros juegos con la primera versión de DIV, no teníamos más remedio que ajustarnos a las posibilidades sonoras del mismo, e introducíamos sonido en el único formato de audio que esta aplicación soportaba. DIV 2 tiene más posibilidades.

Con la llegada de DIV 2, se nos ofrecían una serie de posibles formatos, con nombres ilegibles que no nos aclaraban nada al respecto de sus características. Era en este momento donde se nos planteaba el gran dilema: ¿y qué formato utilizo yo?. Por supuesto, si no disponemos de software para editar en todos los formatos que DIV 2 soporta, no tendremos problemas a la hora de elegir, pues nos tendremos que ceñir a las posibilidades de nuestro programa habitual. Pero si disponemos, por ejemplo, de un completo secuenciador como *ModPlug Tracker*, que permite realizar composiciones en todos los formatos de DIV 2, tendremos que decidirnos por uno de ellos. En este número vamos a hablar en profundidad de estos formatos, para ayudar al lector a tomar esta delicada decisión.

El formato MOD es el más limitado y XM es el formato más potente



ModPlug Tracker nos permite trabajar con cualquiera de los formatos que soporta DIV 2.

Formatos soportados por DIV 2

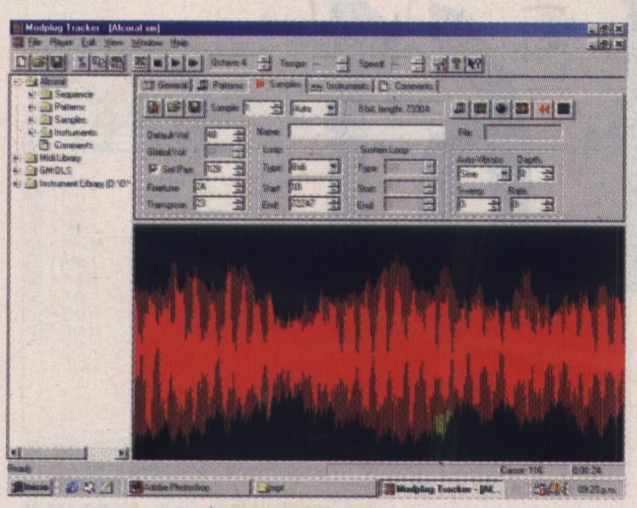
Los formatos soportados por DIV 2 son: WAV, MOD, S3M y XM. El primero es el formato que la primera versión de DIV nos permitía utilizar y es, sin duda, el más rudimentario. Lo utilizaremos para añadir todo tipo de efectos sonoros, incluso cuando utilicemos música comercial, obtenida desde fuentes externas tales como CDs o musicasettes.

Este formato no nos permite hacer nuestras propias composiciones musicales y, además, si deseamos obtener una buena calidad de sonido, se necesita ocupar mucha memoria. Esta última es la principal desventaja del formato WAV, y por este motivo se han incluido los restantes tres formatos en la última versión de DIV, ya que permiten realizar composiciones complejas a partir de WAV que representarán a los instrumentos. Como estos instrumentos serán bastante cortos (basta con tener grabado el sonido de una nota), podrán ser sampleados con una alta frecuencia de muestreo, y podremos realizar entonces composiciones con una gran calidad sonora sin que apenas importe la longitud de las mismas, sino tan sólo los instrumentos utilizados.

El formato MOD

Es el precursor de todos los anteriores, de manera que es el más antiguo. Los programas más extendidos que utilizan este formato son *Protracker* y *NoiseTracker*. Es el formato musical más comúnmente soportado. Este formato ha sido heredado de los primeros secuenciadores que estaban escritos para correr en los legendarios ordenadores AMIGA. De todos modos este es, probablemente, el formato más viejo de ficheros para secuenciadores y, por consiguiente, tiene una gran cantidad de limitaciones. Es muy importante tener en cuenta estas limitaciones si nos disponemos a grabar una de nuestras creciones en este formato, ya que podríamos perder irrevocablemente parte de la información que tanto esfuerzo nos había costado generar. Estas son las limitaciones del formato MOD:

- No hay posibilidad de usar instrumentos, es decir, que los samples que representan a los instrumentos sonaran tal cual han sido sampleados.
- Tendremos un máximo de 31 samples, lo cual limita el número de instrumentos a utilizar en nuestras creaciones.
- El tamaño de cada sample está limitado a 128 k, lo cual significa que no podremos samplear largos fragmentos para introducirlos en nuestras canciones.
- La información de los samples deberá estar codificada con palabras de 8 bits, lo cual reducirá su nitidez.
- Se tendrá un límite de 64 filas por pattern. Hay que tener un especial cuidado con esto. Para darnos cuenta del peligro analicemos una posible situación que podría originarse en una cotidiana sesión de *FastTracker*: acabamos de realizar una estupenda composición con patterns de 80 líneas. Después de deleitarnos con ella por última vez decidimos que ya es hora de irse a la cama. Vamos al menú Disk.Op. y, presa de un cansancio notorio, le damos apresuradamente a grabar sin fijarnos muy bien en lo que hacemos. Nos vamos a dormir con una gran sonrisa de satisfacción. Llega un nuevo día. Encendemos rápidamente nuestro PC y pronto somos testigos de la desagradable sorpresa: *FastTracker* ha truncado



Con el formato XM podremos utilizar samples de 16 bits.

nuestros patterns a 64 líneas y el compás de nuestra maravillosa canción ha pasado de ser un original cinco por cuatro a un vulgar cuatro por cuatro, que además suena francamente raro.
Conclusión: El trabajo de toda una noche y un momento de mágica inspiración se han ido al traste.

- Se tendrá un máximo de 64 patterns diferentes. Este límite rara vez se llegará a sobrepasar, pero en caso de hacerlo, las consecuencias de grabar una canción con un número mayor de 64 patterns diferentes, podrían ser catastróficas.
- Así mismo, la secuencia de patterns (tanto iguales como diferentes), tendrá como mucho una longitud de 128.
- No existe columna de volumen como la que aparece en *ModPlug Tracker*, de manera que si la editásemos no tendría ningún efecto.
- Hay unos cuantos efectos que MOD no permite realizar
- No es posible poner comentarios.
- La velocidad inicial de la canción, así como su tempo y volumen iniciales, tampoco serán almacenados, de manera que si no escribimos estos valores en la columna de efectos, justo al inicio de la canción, se tomarán valores por defecto, lo cual suele originar que la canción comience reproduciéndose a una velocidad no adecuada y con un volumen que nos es el que se desea.
- La panorámica esta fijada por defecto para cada canal.
- No existe la nota de corte de sonido (la que en *FastTracker* se escribe con la tecla Bloq. Mayús).

Nosotros no recomendamos el uso de archivos MOD. Si estás familiarizado con la sintaxis de efectos de este formato puedes editar en formato XM, que dispone de una considerablemente mayor cantidad de efectos y muchas más prestaciones. Además, el formato XM es también bastante popular en PC.

El formato S3M

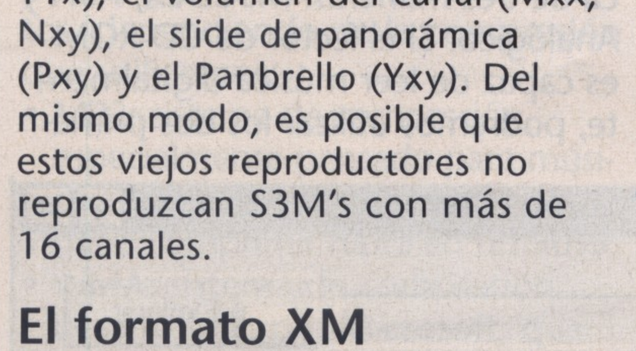
Este formato surgió junto con el programa *Scream Tracker 3*, uno de los secuenciadores más populares en PC. Tiene todas las prestaciones del formato MOD e introduce muchas más. Está bastante limitado aún si lo comparamos con el formato XM o IT, especialmente si tenemos en cuenta que carece de envolventes de volumen, pero podemos decir sin reparos que, pese a esto, es un formato muy potente.

Una de las grandes ventajas de los módulos S3M es que son tan populares como los del formato MOD, y pueden ser reproducidos por cualquier reproductor decente de módulos. La única pega es que, la composición en formato S3M, no nos permite la utilización de instrumentos (que no samples, que sí que son permitidos) y todo lo relacionado con ellos (envolventes, mapeado de notas), y tiene la misma limitación del formato MOD de 64 filas por pattern. Los comentarios de las canciones no serán grabados en los módulos S3M y hay un máximo de 32 canales. Aparte de esto, es un excelente formato, aunque algunos reproductores, los más antiguos, pueden no reconocer los nuevos efectos S3M introducidos por *Impulse Tracker*, como el slide de tempo (T0x, T1x), el volumen del canal (Mxx, Nxy), el slide de panorámica (Pxy) y el Panbrello (Yxy). Del mismo modo, es posible que estos viejos reproductores no reproduzcan S3M's con más de 16 canales.

El formato XM

Este es el formato que introdujo *FastTracker* en el panorama musical informático. El formato XM es como tomar el formato MOD y elevarlo a la enésima potencia. Se puede encontrar todo aquello que hay en los MODs, pero además encontrarás un gran número de prestaciones más. Son las siguientes:

- Un mayor número de efectos.
- Soporte de samples sin límite de tamaño.



Los formatos MOD y S3M no permiten aplicar envolventes.

- Admite samples de 16 bits.
- Número variables de filas por pattern.
- Más de 64 canales a disposición del usuario
- Uso de instrumentos: antes el término "instrumento" y el término "sample" eran prácticamente análogos. En los formatos XM e IT, un instrumento puede tener más de un sample.

El único problema del formato XM es que los samples no pueden ser compartidos por instrumentos diferentes. Por eso, si queremos tener, por ejemplo, un mismo sample de guitarra almacenado en dos bancos diferentes, con el fin de que uno de ellos suene siempre por el canal izquierdo y el otro por el derecho, no tendremos más remedio que cargarlo dos veces, haciendo que se incremente el tamaño del archivo de nuestra canción en el doble del tamaño de ese sample de guitarra y preservando así la compatibilidad con el formato XM.

Otro pequeño inconveniente, que sólo se da con *FastTracker*, es que éste programa no puede cargar módulos con número de canales inferior a 4, ni tampoco superior de 32. En caso de querer editar canciones en formato XM con estas características deberíamos utilizar *ModPlug Tracker*.

S3M es más completo que MOD, pero no tanto como el formato XM

La principal ventaja del formato XM sobre los módulos de *Impulse Tracker* es que están más extendidos en PC que los del formato IT, que no son soportados por muchos reproductores existentes.

Conclusiones

Ahora ya conocemos las características de los diferentes formatos sonoros que soporta DIV 2. A la vista de los datos aquí reflejados parece claro que la mejor elección sería la de realizar la música en formato XM, que parece ser el más potente, y los efectos sonoros en formato WAV. De todas maneras, es posible que algunos de los lectores estuvieran previamente familiarizados con alguno de los otros formatos, sobre todo en cuanto a la sintaxis de los efectos se refiere. De ser así, recomendamos que cada uno se exprese en el formato que más cómodo le parezca, ya que de poco serviría utilizar un formato muy potente si no sabemos explotarlo al cien por cien.

Sergio Cánovas

Audiograbber 1.61

Grabación digital de CD-Roms

En esta edición vamos a echarle un vistazo al programa Audiograbber v1.61, una versión recientemente aparecida en el mercado. Este programa es capaz de leer la información de un CD de música y, sin pérdida ninguna de información, hacer una copia exacta en nuestro disco duro.

Empezaremos por una breve introducción explicando la estructura de un CD, cómo está distribuida la información y por qué es más fiable la copia de la música de esta manera y no a través de la tarjeta de sonido.

Un CD-ROM está dividido en sectores de 2352 bytes, de los cuales 2048 se utilizan para el almacenamiento de datos; el resto de ellos se utilizan para la sincronización (principio y final de la información y número de sector). Gracias a esto al ordenador le resulta muy fácil leer cualquier dato que necesite. Sin embargo un CD de música utiliza los 2352 bytes para el almacenamiento de información. Al ordenador le resulta muy difícil saber dónde empieza y termina un sector (todos ellos son como una larga secuencia de bytes sin principio ni fin), por lo que, si queremos leer un sector específico, puede ser que no podamos.

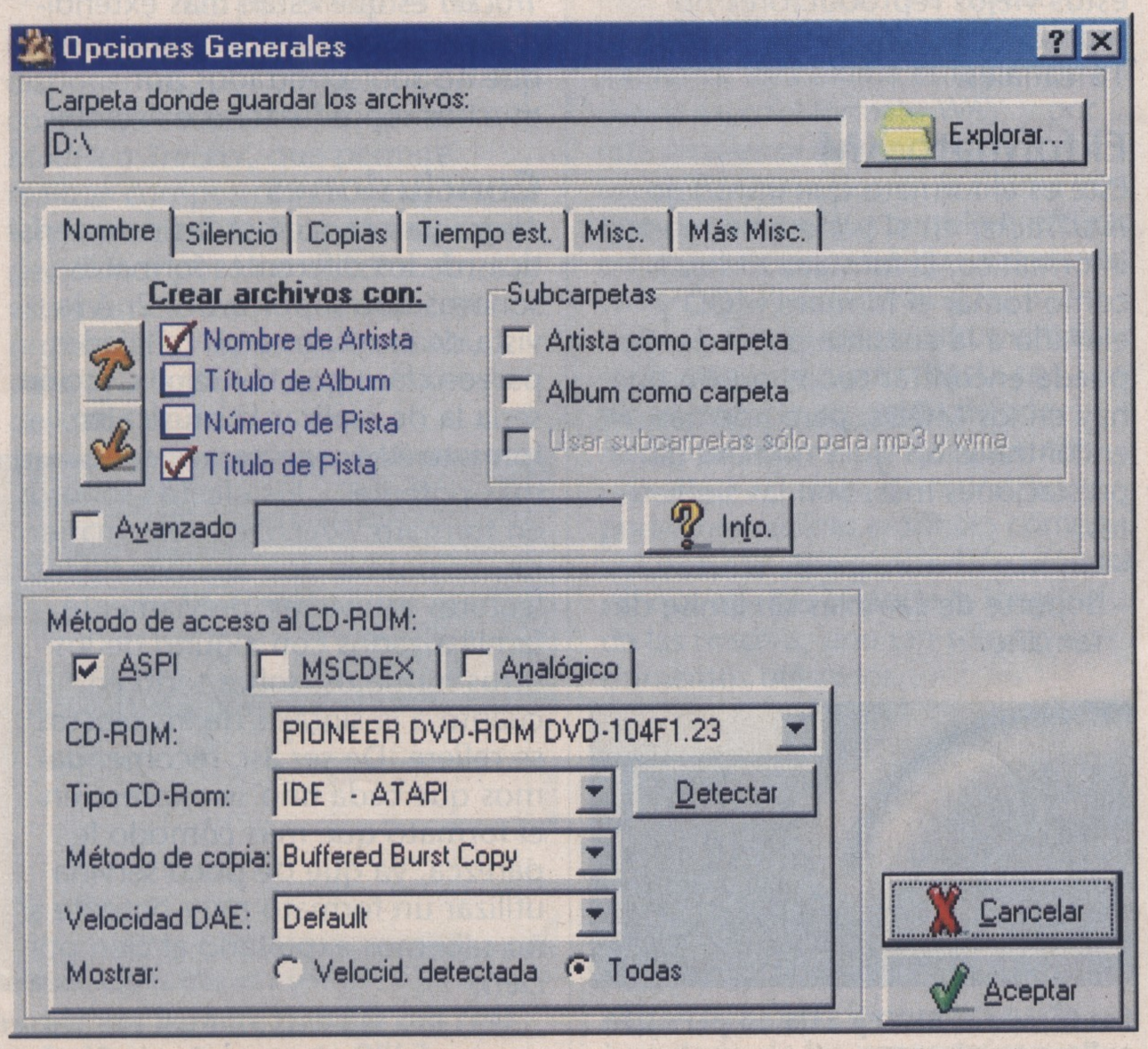
El programa puede grabar música de tres maneras: MSCDEX, ASPI y Analógico. Si el lector de CD-ROM es capaz de leer música digitalmente, podremos utilizar los dos primeros métodos, si no, no nos quedará mas remedio que utilizar el tercero, con la consiguiente pérdida de calidad (Ya que la información pasa del lector de CD-ROM (digital) a la tarjeta de sonido (analógico) y luego al disco (digital).La velocidad de grabado que se consigue con este método es de 1X.

Si tenemos un CD-ROM más moderno, podremos utilizar uno de los otros dos métodos, que voy a describir a continuación.

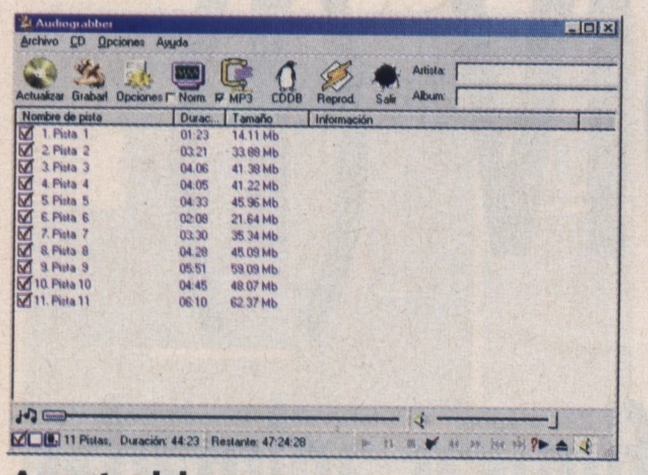
MSCDEX: es un sistema antiguo, que utiliza un buffer intermedio donde almacenará la información que lea del CD, sin embargo, cuando deja de leer (porque esté el buffer lleno o porque se haya acabado la información) será muy difícil sincronizar el nuevo acceso al CD exactamente donde lo había dejado. Esto lo arregla Audiograbber leyendo más datos de los que escribe a disco, para así poder comparar el principio de la nueva lectura con el final de la anterior ("llamado overlapping", "sincronización" o "corrección jitter"). La velocidad con este método depende del número de sectores que se lean de una sola vez y del tamaño del overlap.

ASPI : utilizado en los lectores más modernos, este sistema es capaz de leer información sin parar (sin pérdida de sincronización) a la vez que se va escribiendo al disco, por lo que no se pierde tiempo en la sincronización. La velocidad de este sistema de grabación viene determinada por la velocidad de lectura del lector CD-ROM y la velocidad del ordenador. Si el lector de CD-ROM es más rápido que lo que el ordenador puede manejar puede haber fallos de sincronización.

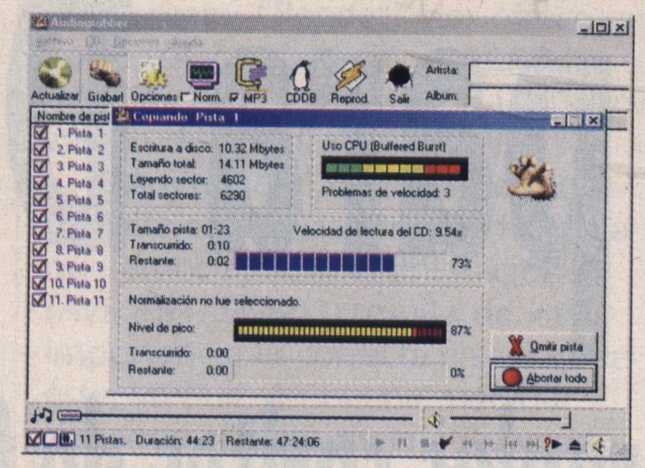
Una vez hecha esta breve introducción pasemos al programa en sí. Audiograbber es capaz de leer música de un CD digitalmente y pasarla a disco (utilizando el formato y compresión que más nos guste: MP3, WMA, WAV,...) Todo depende de los "codecs" que tengamos instalados en el sistema y las librerías que hayamos añadido al programa (están en



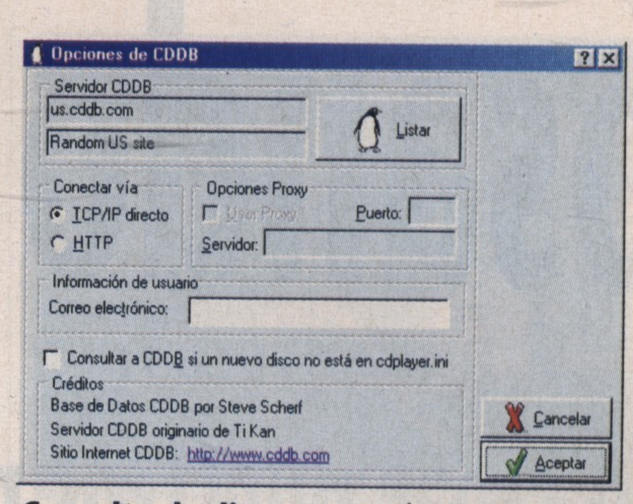
Opciones generales del programa.



Aspecto del programa.



Grabación de una pista.



Consulta de discos a través de Internet.

continua expansión). Además tiene la opción de grabar música por la entrada de audio de la tarjeta de sonido (muy recomendable para las tarjetas con entrada digital de audio, donde se conectará el miniDisc o cualquier otro sistema de grabación digital, como Sound Blaster 1024).

Instalación y configuración

El programa se instala automáticamente, sin ningún tipo de complicación (tanto en su versión shareware como en la versión completa.) La diferencia entre la versión de prueba y la completa es que, en la versión de prueba, no se pueden grabar más de la mitad de canciones, elegidas aleatoriamente.

Una vez instalado, vamos a hacer un breve repaso del programa: en un principio podemos ver una pantalla donde se nos muestra toda la información que se posee del disco insertado en el lector. Información que se saca de dos fuentes diferentes: cdplayer.ini y CDDB.

- Cdplayer.ini: archivo que se encuentra en el disco duro y que posee la información que hemos suministrado al reproductor de CD de Windows en algún momento. (como el intérprete de una canción, el nombre de las pistas, el tipo de música...)
- CDDB: (Compact Disc Database) Es una base de datos en Internet que tiene información sobre CDs de música (información que ha sido suministrada por todo el mundo a lo largo de los años) en la cual se puede meter información sobre un disco, o sacarla.

Opciones generales: para ponerle nombre a los archivos de las pistas grabadas se nos ofrece una amplia gama de posibilidades: con toda la información del CD podemos hacer combinaciones; por ejemplo: Blind Guardian - Nightfall in Middle Earth - Mirror mirror.mp3 (Autor - Disco - Canción.). También deberemos seleccionar el tipo de lector que poseemos y el tipo de lectura que deseamos realizar, así como la velocidad de lectura y distintos parámetros que dependerán tanto del lector como del disco que va a ser grabado.

Normalizado: al grabar un CD, tenemos la opción de ajustar el volumen con que se grabará la canción, a esto se llama normalizar. Cuando el nivel de pico (volumen máximo de la grabación) llega a un valor que nosotros proporcionaremos, el volumen de la grabación de incrementará o descenderá.

Opciones MP3: cuando se graba una canción desde un CD se puede guardar en distintos formatos, dependiendo de la compresión que utilicemos.

- MP3: sistema de compresión, especialmente pensado para música con calidad de CD. Puede llegar a comprimir hasta en un 90%.
- WMA: sistema de compresión desarrollado por Microsoft, que en algunos casos puede comprimir hasta en un 95%.
- WAV: sistema de grabación de sonido que no comprime los datos; éstos son guardados en formato PCM (Pulse Code Molulation) que es el formato que utilizan los CDs de música.

Para la compresión en formato MP3 se nos da la opción de utilizar algún programa externo para la compresión o un "codec" o librería. En la página de Audiograbber podremos encontrar la librería "BladeEnc.dll", una de las pocas librerías de compresión MP3 totalmente gratuitas.

Grabación

A la hora de grabar un CD deberemos introducirlo en la unidad, seleccionar las pistas que queramos grabar, asignarlas nombres, así como al disco y darle a la opción Grabar. Si además queremos pasarlas a MP3 seleccionaremos la opción MP3 y si queremos normalizarlas, lo mismo. Nos aparecerá una pantalla como la de la figura, en la cual podremos ver todos los detalles del proceso de grabación.

En un alarde de imaginación sin limites hemos llamado a los bandos "los buenos" y "los malos"

Grabación por entrada de línea

Una opción curiosa es la grabación de la entrada de línea de la tarjeta de sonido. Podremos acoplar el miniDisc o cualquier aparato de reproducción de sonido digital y grabar las canciones a nuestro disco duro sin pérdida de calidad.

Tendremos la opción de dividir en pistas manualmente o automáticamente, así como programar la hora de comienzo y final de la grabación (muy útil si queremos grabar de la radio.)

Para terminar

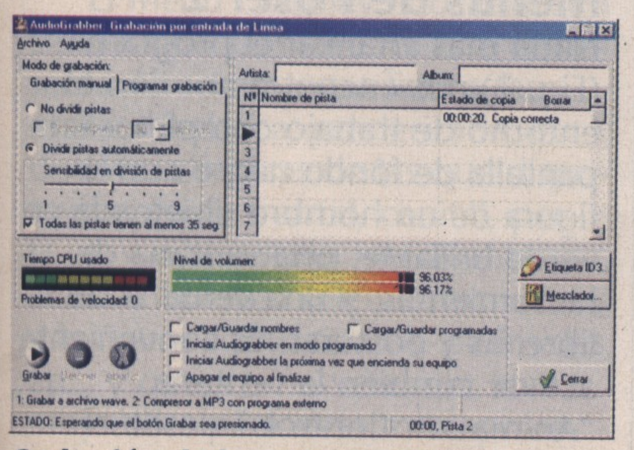
Si queréis más información sobre Audiograbber, los datos son:

Autor: Jackie Franck

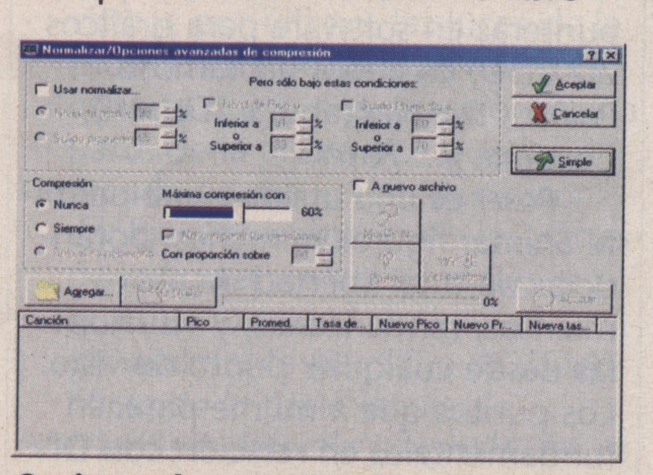
Web: http://www.audiograbber.com-us.net y http://www.dezines.com/audio

Hasta el próximo número de la revista. Espero que os haya sido útil.

Javier Fernández



Grabación de la entrada de línea.



Opciones de normalización de canciones.

Glosario de términos

Codec: algoritmo de compresión-descompresión.
Librería: conjunto de funciones o rutinas que realizan algún tipo de tarea.
Buffer: porción de memoria que se utiliza para el intercambio de información entre dos dispositivos.

Posando para DIV (I)

Cómo crear nuestros propios sprites de figuras animadas

Cuando leía las primeras informaciones sobre la aparición del nuevo DIV2 había una cosa que me resultaba particularmente atractiva: el Generador de Sprites. ¿Cómo, pensaba, podrán hacer un generador de sprites que se adapte a cualquier figura, postura, textura, o juego que se programe?

La aparición de DIV2 vió confirmadas mis sospechas: Hammer había tirado por la calle de en medio para ofrecernos un Generador de Sprites ciertamente limitado.

El Generador de Sprites de DIV2 tiene tres desventajas claras:

– Las figuras que vienen no pueden modificarse a nuestro gusto (sólo hay un hombre, una mujer y un niño, que por medio de texturas deben transformarse en orcos, esqueletos y demás fauna), ni añadirles o quitarles atributos (si queremos un pirata al que le falta una mano y lleva un gancho a cambio, hay que hacerlo sprite a sprite después de creados éstos, con el enorme trabajo que conlleva).

Poser es una utilidad que permite animar figuras humanas, dotándolas de total libertad de movimientos

– Las posturas y acciones que traen incorporadas no son modificables. Esto es, si yo quiero hacer un personaje que haga un "mate" de baloncesto, lo deberé hacer a mano.

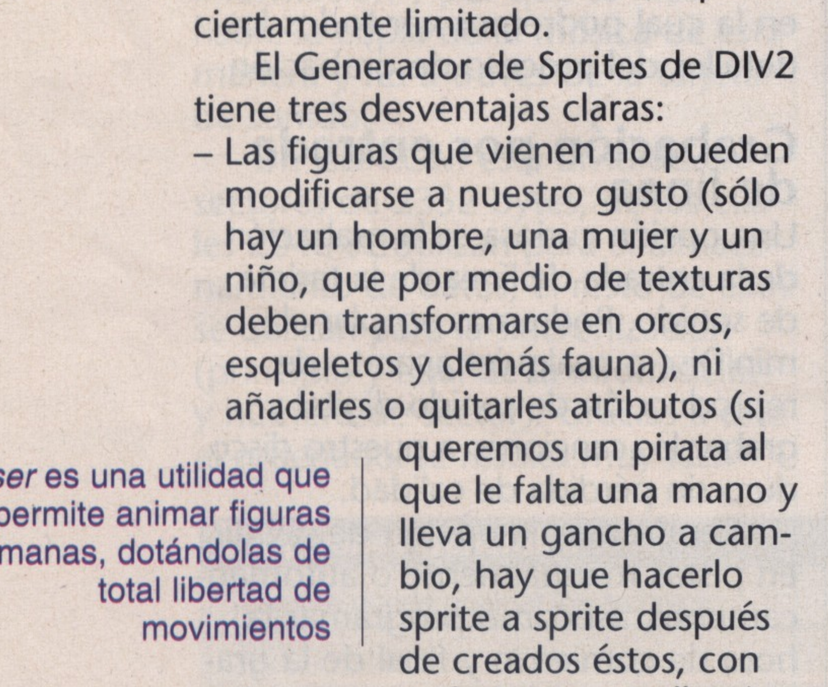


Fig. 2 - Exportando a Div.jpg.

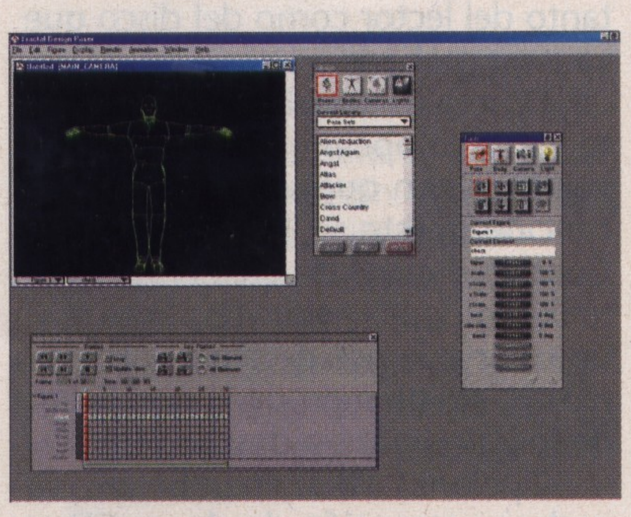


Fig. 1 - Pantalla inicial de Poser.

– No se pueden incorporar objetos a las figuras existentes. Es cierto que el Generador de Sprites incorpora figuras "armadas", que lo único que hacen es mostrar un palo entre sus manos y que, como dice el manual, deberán modificarse posteriormente en el editor gráfico (sprite a sprite, claro...).

Por tanto, a menos que queramos hacer solamente hombres, mujeres o niños en nuestros programas (eso sí, armados de palos), debemos buscar otra solución a nuestros problemas. Y esta solución nos llega de la mano de Metacreations, una de las empresas punteras en software para gráficos 2D y 3D, creadora del famoso *Painter, Detailer* (ahora *3Dpaint*), *Canoma*, y... *Poser*.

Poser es una utilidad que permite animar figuras humanas, dotándolas de total libertad de movimientos, renderizarlas, y visualizarlas desde cualquier punto de vista. Los puntos que a mí me parecen fundamentales en relación con DIV son éstos:

– Incluye muchas más figuras que el Generador de Sprites (hombre y mujer de negocios, hombre, mujer y niño informales, hombre, mujer y niño desnudos, esqueletos de hombre y mujer, esos juegos de rol van a tener ya sus esqueletos luchadores, incluso maniquíes de madera).
– Estas figuras son totalmente articulables. Cualquier postura que queráis representar la podéis conseguir en pocos segundos, y uniendo varias posturas podréis editar vuestras propias animaciones.
– Las figuras son totalmente configurables. Si queremos que ese esqueleto a animar porte una lanza y un escudo en las manos, que lleve casco y botas, y que a su alrededor sobrevuele un murciélago a la vez... ¡lo podemos hacer!

En la actualidad se encuentra en su versión 4, y permite crear incluso ¡expresiones! en la cara de la figura a animar. Sin embargo, para nuestros propósitos nos centraremos en la versión 2 de *Poser* por dos razones fundamentales:

1) Cubre sobradamente nuestras necesidades para Div y es capaz de dotar a nuestro próximo juego de ese aspecto profesional que tanto ansiamos.
2) Apareció una versión gratuita ¡completamente operativa! en el número 22 de la revista 3D World de esta misma editorial Prensa Técnica, argumento que creo convencerá a todos de la idoneidad del producto.

Un recorrido por los menús de Poser 2

Nada más arrancar el programa (Fig. 1) nos encontramos con el entorno de trabajo completo: una pantalla de fondo negro con la figura de un hombre silueteada en verde brillante, una ventana de herramientas, y una ventana de librerías y posturas. Es conveniente activar también la ventana "Animation Controls" desde el menú "Windows", aunque serán

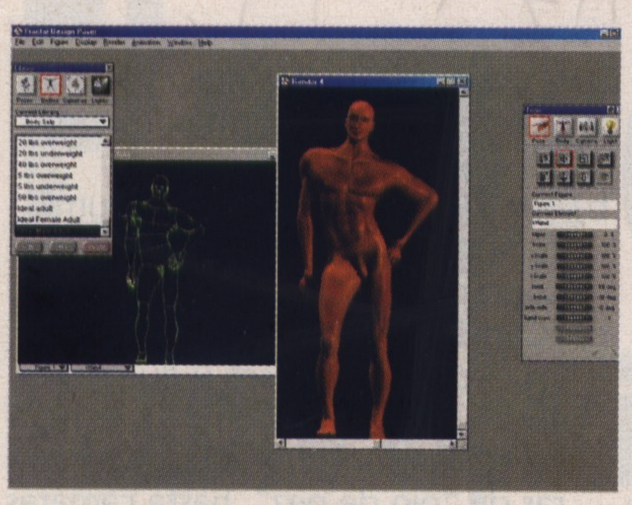


Fig. 3 - Podemos crear las postura.

comentadas en capítulos posteriores de este curso.

En el menú "File" podemos comprobar como es posible importar y exportar objetos de *3d Max, 3d Studio, Autocad, Wavefront,* etc... lo cual nos será de mucha ayuda en nuestros próximos pasos.

El menú "Figure" es quizá el más importante de *Poser*. En él podemos seleccionar las distintas figuras que queramos incluir en la animación, sus atributos (aunque esto veremos luego cómo configurarlo a nuestro gusto), el programa incluye unas cuantas opciones: más musculoso, gordo, delgado, alto, bajo, adolescente, niño, adulto; tipos de manos que queremos (se pueden modificar a lo largo de la animación, así que podemos representar un hombre que se acerca con la mano estirada a empuñar su espada, y como, una vez asida ésta, la mano se transforma en un puño); el tipo de Cinemática Inversa que deseamos (ver cuadro 1); si deseamos utilizar límites (se refiere a las articulaciones, ya veremos la necesidad de aplicar límites al giro de aquéllas para impedir la creación de posturas "contorsionistas", ¡aunque si las necesitamos podemos crearlas!); si queremos aplicar simetría o no a la postura (si quiero que la pierna izda. se comporte exactamente igual que la derecha, etc.), lo cierto es que yo no lo aplico nunca por el carácter antisimétrico del cuerpo humano; y, por último, opciones para añadir más figuras, eliminarlas, y ocultarlas (muy útil cuando tenemos cinco figuras en pantalla, y *Poser* comienza a volverse muy lento).

El menú "Display" nos muestra las distintas posibilidades de representación de nuestra figura (en malla alámbrica, renderizada, etc.), distintas cámaras de las que disponemos (vista frontal, cenital, perfil...), y una opción interesante: insertar "props" (elementos ajenos a la figura como esferas, pirámides, etc.; y que, como veremos, son muy útiles para desarrollar posteriormente la animación).

El menú "Render" nos ofrece un corto, pero suficiente, surtido de opciones de renderización y texturización de nuestra animación.

"Animation", por último, nos permite especificar en qué formato queremos realizar la animación (AVI, TIFF, BMP) y resulta de vital importancia la correcta elección de éste, ya que, en buena medida, depende el éxito de la utilización posterior que vayamos a hacer desde DIV de nuestra figura animada. También nos permite decidir de cuántos "frames" (cuadros o viñetas) va a constar nuestra animación.

La pantalla principal de Poser

Una vez visto el sistema de menús de *Poser*, pasemos a la pantalla de trabajo y las distintas ventanas que nos aparecen.

La pantalla de la figura nos muestra la misma desde la vista que hayamos seleccionado en el menú "Display>Camera View", y que pueden ser las típicas de cualquier programa 3D: vista de frente, izquierda, derecha, desde arriba, o una vista que hayamos creado nosotros desde la opción "cámaras" que comentaremos en unos momentos.

En esta pantalla es donde veremos la postura que está tomando nuestra figura con las transformaciones que hagamos. También podremos modificar en ella cualquier parte de la anatomía que queramos, pero el consejo mejor que se puede dar para el uso de *Poser* es: no utilices esta ventana (salvo en contadas excepciones, como es el situar los pies de la figura) para cambiar la postura, utiliza el control de animación y sus "rodillos" de control.

En la parte de abajo de la ventana de la figura veremos dos pestañas desplegables:

– La primera nos permite seleccionar, en el caso de que hayamos incluido más de una figura en la pantalla, la figura deseada.
– La segunda nos permite seleccionar la parte de la figura que deseemos, el cuerpo entero, los props que hayamos creado, la cámara, vista y, por último, las luces de la escena. Esta pestaña es muy útil cuando queremos seleccionar una parte de la figura que queda tapada por otra. Hay que recordar que los nombres de las partes de la figura vienen en inglés, y que los nombres comienzan generalmente por "l" o "r", identificando la parte izquierda ("left") o derecha ("right"). Así, si en la pestaña desplegamos y

Esqueletos para nuestros juegos.

escogemos "lThigh" se resaltará en la figura en color blanco el muslo izquierdo, puesto que "lThigh" corresponde a "l" de "izquierda" y "Thigh" de "muslo" (en inglés, claro).

La ventana "Tools" nos muestra las distintas opciones que tenemos para crear la imagen adecuada para nuestra figura: "Pose" nos permite alterar los distintos componentes anatómicos de la figura, "Body" altera el conjunto del cuerpo como una unidad, "Cameras" proporciona las herramientas para crear nuestras propias cámaras y animarlas y, por último, "Lights" permite crear la ambientación lumínica adecuada de cara al render final de nuestra escena.

El Generador de Sprites de Poser es mucho más completo que el que venía en DIV 2

De este modo, una vez seleccionemos una opción u otra, las herramientas que comentamos ahora afectarán exclusivamente a la opción escogida. Dicho con otras palabras, si selecciono "Body" y posteriormente el icono de "Rotate", lo que me permite esta herramienta es rotar todo el cuerpo. Si posteriormente selecciono "Pose", me permitirá rotar una parte del cuerpo.

Bajo las opciones generales podemos encontrar los iconos correspondientes a las herramientas para modificación sobre la ventana de la figura: "Rotate" (rotar),

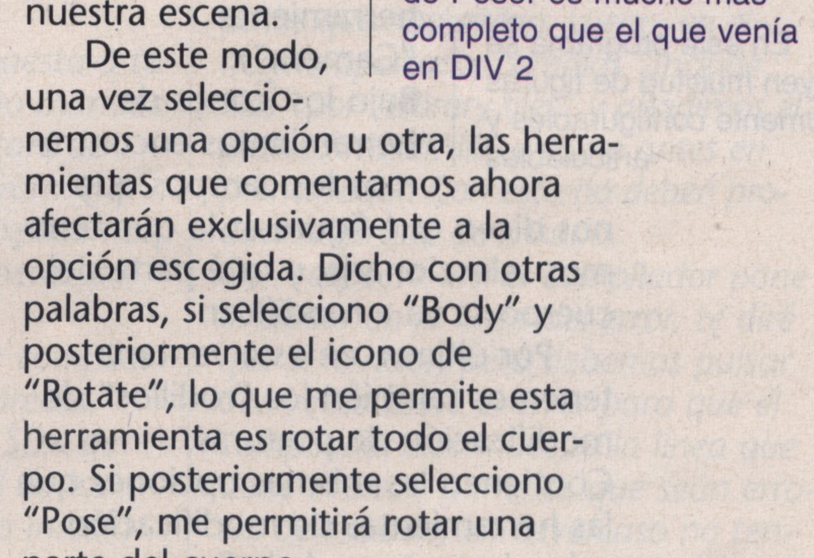


Fig. 4 - Ya tenemos nuestra heroína.

Un poco de cinemática

La Cinemática Inversa es un método de animación que se opone a la Cinemática Directa tradicional. En éste método, si queremos hacer que un personaje estire la mano para agarrar algo sobre una mesa, debemos primero mover el hombro del personaje, luego su brazo, su antebrazo y, por último, su mano para poder animarlo totalmente. Así, hemos movido primero los elementos "padre" (el hombro es jerárquicamente superior al brazo, y éste al antebrazo, y así sucesivamente) para llegar hasta el "hijo" (la mano).

En la Cinemática Inversa, es la mano la que "tira" de toda la jerarquía (al mover la mano, movemos el antebrazo, y éste tira a su vez del brazo, etc.). Este método es el más utilizado en la actualidad por todos los animadores 3D, ya que proporciona una naturalidad a los movimientos que no se consiguen con la animación directa.

"Translate" (mover), "Scale" (escalar), "Chain Break" (romper enlace: permite mover una parte de un cuerpo, manteniendo fija las partes a las que está unida; si rompo el enlace entre el antebrazo y el brazo, podré mover la mano y el antebrazo en cinemática inversa, que el brazo seguirá fijo en su sitio), "Twist" (girar), "Ztranslate" (mover en el eje Z), "Taper" (afilar: aplicado a una parte del cuerpo hará que la zona superior sea más ancha que la inferior, o a la inversa) y, por último, "Focal Length" (Largura Focal, permite cambiar las lentes de las cámaras; esta opción sólo está disponible al activar la herramienta "Camera").

En este programa se incluyen multitud de figuras totalmente configurables y articulables

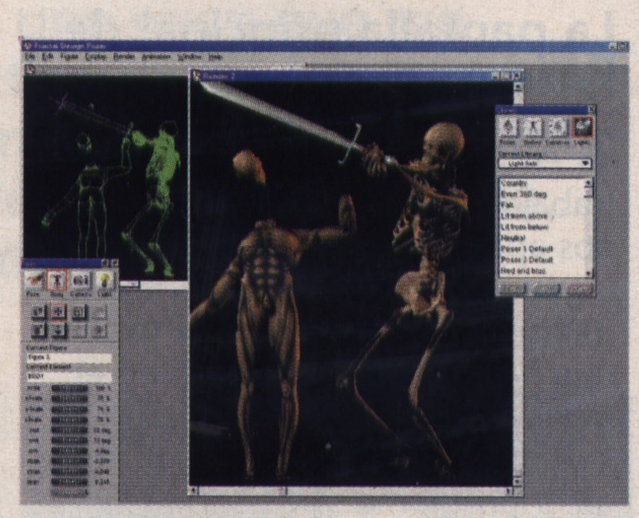
Bajo los iconos de herramientas encontramos los campos que nos dicen qué figura es la que tenemos seleccionada, y qué parte del cuerpo se va a modificar.

Por último, en esta ventana tenemos también los "rodillos" de modificación de posturas. Contienen las mismas opciones que las herramientas de modificación sobre la ventana de la figura, aunque más ampliadas, y lo que es más importante: permiten un control mucho más exacto que al hacerlo "a mano". Estos controles variarán en cada momento según la selección que hagamos: no serán iguales los controles cuando seleccionemos la cintura de la figura, que cuando seleccionemos toda la figura, o una cámara.

La biblioteca de Poser

Otra ventana que aparece en el espacio de trabajo de *Poser*, es "Library", o biblioteca predefinida de posturas, animaciones, etc. En ella, vemos cuatro opciones:

a) "**Poses**": posturas predefinidas agrupadas por temas, como deportes, carrera, acción, etc. Si pulsamos "New " nos permite crear nuestra propia agrupación en la que iremos almacenando las posturas que formarán nuestra animación.
b) "**Bodies**": tipos de figura humana: desde 10 hasta 50 libras de sobrepeso, adultos ideales, etc.
c) "**Cameras**": cámaras creadas por los expertos de *Poser*, y que nos ahorrarán mucho tiempo a la hora de crearlas nosotros. Desde cámaras de "ojo de pez", hasta cámaras cenitales, desde abajo, etc.
d) "**Lights**": luces predefinidas que permiten simular cualquier situación: desde cualquier estación del año ("Summer", etc.) hasta luces que abarcan los 360º de la figura.



Podremos animar nuestras escenas.

Este ha sido un recorrido rápido por las distintas opciones de Poser. En próximos capítulos crearemos una figura propia, a la que añadiremos accesorios, crearemos texturas para ella, animaremos, y por último, exportaremos a Div. Si queréis realizar alguna consulta, podéis hacerlo a tayete@teleline.es

Santiago García Mazariegos "Tayete"

Consejos

Unos consejos básicos para aquellos que se animen a ir probando las posibilidades de *Poser*:

1) Antes de comenzar a animar, pensad muy bien cómo va a ser vuestra figura. *Poser* toma la figura que animemos independientemente en cada postura de la animación, por lo que, si a mitad del proceso, decidimos hacer un cambio en la figura, habría que hacerlo posteriormente postura por postura en la animación. Si en la mayoría de las imágenes del programa vuestro personaje aparecerá con los puños cerrados, poned así las manos en vuestra figura. Si sus proporciones son distintas a las que vienen por defecto, modificad esto antes de nada.

2) Activar la Cinemática Inversa para las piernas pero no para los brazos (el modo en que *Poser* maneja las articulaciones de brazos y piernas varía).

3) Procurad animar usando los controles de animación independientes, y no las partes de la figura. Por ejemplo, para rotar la cintura de una figura no seleccionéis "waist" (cintura) y luego la herramienta "Rotate" y lo hagáis girar "a mano"; en su lugar seleccionad "waist" y, en el control de animación, moved el rodillo correspondiente a "Rotate", permite mayor control y precisión.

4) Situad un prop que os indique siempre cuál es el centro de coordenadas de *Poser*. Yo suelo situar un cono (menú Display>Add Prop>Cone) en el lugar que sale por defecto en el suelo, para poder alinear siempre la pelvis de nuestra figura con ella y que, al cambiar de postura, la figura no se nos vaya a la derecha o a la izquierda del cuadro.

5) Activad siempre la línea de horizonte (menú Display>Guides>Ground Plane) mientras estemos creando posturas, para poder situar correctamente nuestra figura sobre el suelo.

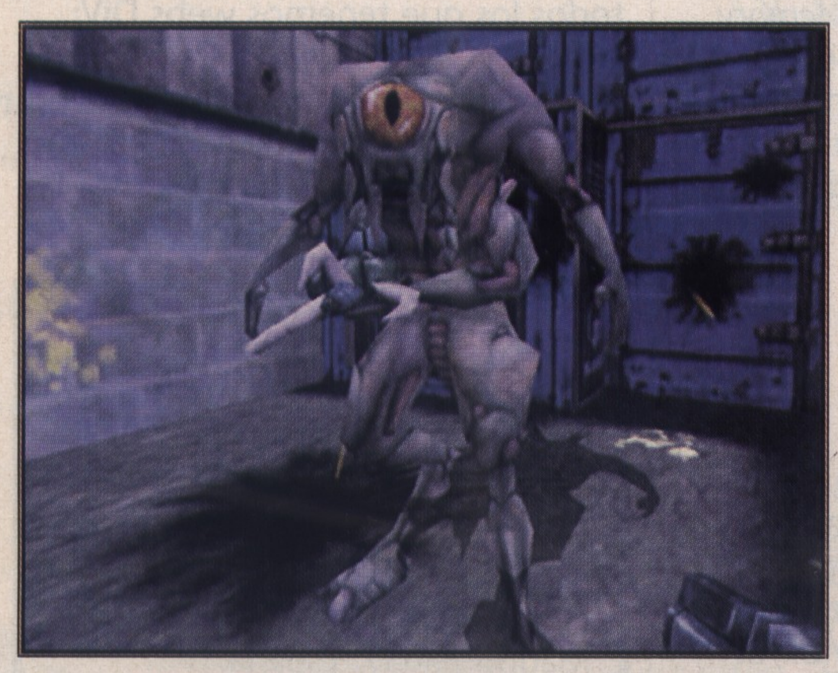
6) La combinación de teclas de "Undo" (deshacer): Ctrl+z, es vuestra mejor aliada. Si algo no te parece que haya quedado bien, deshazlo antes de mover nada más.

7) Y por último, tened siempre documentación a mano sobre la postura a crear. Si quiero hacer una animación de un "mate" de baloncesto puedo hacerlo de memoria, pero siempre quedará más natural si tengo una secuencia fotográfica de los distintos pasos que da el jugador hasta encestar.

El correo del lector

Vuestro espacio

Todas las dudas que podáis tener acerca de la programación de juegos serán raudamente contestadas en este espacio. Así que no lo dudéis, explicarnos vuestras dificultades y pronto dejarán de serlo. También recogemos, como es ya habitual, todas aquellas ofertas de trabajo que puedan interesar a todos los que estén inmersos en el mundo de la programación de videojuegos.

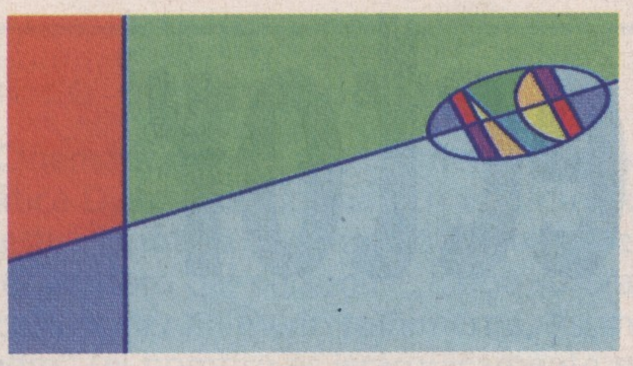


Problemas con Direct X

Hola soy Chema. Para Armando Velez. Tengo una serie de dudas respecto al tutorial de Direct X que impartís en Divmanía. He instalado el programa de ejemplo de la revista y me da errores de compilación. No sé que es lo que hago mal. ¿Puedes decirme qué pasos debo realizar para que el programa se compile y ejecute bien, como la configuración del Visual C++ o el linkado de las librerías?
Cuando se compila con Div2, el propio compilador pone el cursor en la línea donde está el error, Visual C++ no lo hace. Cada vez que compilo me da varios errores, pero no dice en qué líneas, así que tengo que revisar todo el programa, el cual está bien tecleado.
¿Cómo se selecciona, en las opciones del compilador, el directorio del SDK?
Si no te importa me gustaría que me dijeses paso a paso como se configura el Visual C++ y se introduce el programa de ejemplo de la revista. Estoy programando en Div2, pero quiero aprender C++ y utilizar las Direct X para poder desarrollar algún juego. Somos un grupo de desarrollo de juegos que no queremos quedarnos sólo con el lenguaje de Div.
Gracias por todo. Salu2 Chema.

Saludos. Aquí tienes la respuesta a tu pregunta. Bueno, aunque no mandas los errores que te da, intentaré de todas formas responderte, vamos a ello (por cierto, esto ya lo explicamos en uno de los primeros números del curso, pero en fin):
En primer lugar, para poder compilar algo programado con las DirectX, debemos tener instalado el SDK de las Direct, concretamente el de las Direct 6 ó 7. El SDK no es la instalación que viene para ejecutar los juegos, sino un paquete de desarrollo con documentación y ejemplos. Debemos tener también el Visual C++ 6.0, o la versión 5.
Ahora hemos de decirle al Visual que coja las librerías de DirectX, y para eso

nos vamos a "Tools-Options", y en la pestaña "Directories", en la opcion "Include files", añadir una entrada que llegue al path donde hemos instalado el SDK. Por ejemplo, si lo hemos instalado en c:\mssdk, pondremos c:\mssdk\Include y, además, hemos de ponerla por encima de los demás paths que aparezcan. Luego, en el mismo apartado, cambias "Include files" por "Library files" y añadimos el path c:\mssdk\lib, y como antes en primer lugar. Con esto no deben producirse errores de linkado.
Con respecto a si el compilador pone el cursor en la línea del error, te diré que sí lo hace, pero debemos pulsar dos veces sobre el error para que el compilador nos muestre la línea que tiene el error, a menos que sean errores de linkado, en cuyo caso no tendremos más información que la que nos ha dado el linker.
Otra cosa, debemos decirle al compilador que linke las librerías de DirectX, normalmente suelo incluir una sentencia pragma, que es una orden del compilador que le indica



que debe utilizar las librerías, en realidad no hace falta si se la añadimos a las opciones de linkado del proyecto, pero de esta forma, si perdemos el proyecto, y sólo tenemos los fuentes, no seremos capaces de linkar el proyecto. Para ponerlo en las opciones del proyecto, abre "Project-Settings-Link", y en "Object/Library modules", al final de las librerías, pon "ddraw.lib". También puedes añadirlas directamente al proyecto, como si fuera un fichero más, con ello debe compilar el código sin problemas.
Si tienes aun problemas, mándame el proyecto, y la versión del compilador y del SDK. Bye:)

Otra duda

Tengo un problema: cuando creo una instalación en DIV2, o bien el fichero de instalación no funciona, o cuando funciona, el archivo que instala tampoco arranca. Me sale un cuadro de diálogo que me dice que se ha ejecutado una acción no válida y que consulte a la ayuda para ejecutar archivos en MS-DOS (por supuesto, no pone gran cosa). He pensado que tendría que ver con la *Div32run.dll*, pero cuando intento abrir la *dll* con el juego en cuestión, me sale el mismo problema. ¿Qué puedo hacer? No sé muy bien cómo van las *DLLs*.

La DLL en cuestión, que no es otra que DIV32RUN.DLL, es el corazón de DIV en sí mismo. El ejecutable que se crea se encarga de llamar a las distintas funciones dependiendo del código del programa. Todas estas funciones están dentro de esta DLL. Dependiendo de la versión de DIV, la DLL varía, ya que no usan la misma versión DIV1 y DIV2. Por lo tanto esto no tiene nada que ver con tu problema ya que se utiliza en los ejecutables finales. Lo mas seguro es que DIV2 esté mal instalado en tu disco duro, intenta copiar todo otra vez, y sobre todo el directorio INSTALL dentro de donde tengas DIV2.

Anuncios

Pitus Arts/Llach Estudios, estudio freelance de programación fundado en 1992, busca para el apoyo a sus proyectos colaboradores externos: (Ref: Colex001) Se precisa cualquier tipo de ayuda, una vez puestos en contacto veremos qué posibilidades hay y decidiremos si es útil. Se buscan grafistas 2d/3d, guionistas, músicos, programadores en cualquier soporte, etc. Cualquier interesado, que se dirija a llachestudios@worldonline.es
David Llach, Programador Jefe

Lince Games (miembro Stratos), es un grupo en plena formación. Aunque, sobre todo, necesitamos a grafistas y músicos, también son bienvenidos programadores de apoyo, guionistas, diseñadores y todo tipo de ayuda. El juego en proyecto es *Secret of Lost Memory*, programado en DIV 2, a 640x480 pixeles, en perspectiva isométrica. Los interesados deben llamar al 917 333 246 (sólo tardes), escribir a:
Lince Games
C/ Cañaveral, 103, 1ºC.
28029 Madrid
o mandar un mail a:
lincegames@mixmail.com

Guerrero Omhega busca a gente que esté interesada en colaborar en la creación de varios proyectos y la ampliación de esta compañía. Los interesados deben escribir a:
Guerrero Omhega
C/ Europa, nº3 6º segunda
Sant Feliu de Llobregat
08980 Barcelona. España.

Y una aclaración

Salu2
A ver si en la próxima revista se puede decir algo sobre la dirección de ViTAL-web. Aunque inicialmente efectivamente la url era:
http://www.geocities.com/timessquare/arcade/9520,
actualmente está situada en el servidor de pago:
http://www.electronic-soft/vitalweb/
Lo mejor será que se diga el redireccionador, mucho más fácil de utilizar, que es como lo tenemos casi todos los que tenemos webs DiV:
http://pagina.de/vital o
http://pagina.de/vital-web
Además en la de geocities no está la última versión de la web, está la de octubre nada más. Salu2.

Esperamos vuestros mensajes

Para cualquier sugerencia, duda o aclaración podéis mandar vuestras cartas o correos electrónicos a la siguiente dirección:
Divmanía
C/ Alfonso Gómez 42. Nave 1-1-2
Madrid 28037. España.
Tfno: 91. 304. 06. 22
Fax: 91. 304. 17. 97
E-mail: *gover@prensatecnica.com*

Fe de errores

¿Ke tal todo? Te escribo por que he visto la revista, un amigo mío se la kompra siempre y me parece que ha quedado estupenda, muy buena idea la de incluir una recopilación de los juegos que os han ido mandando en DIV, me ha gustado mucho, pero he visto un error, resulta que tenéis dos juegos míos publicados, uno denominado *Navidad* y otro *Super Yohsy*, pero al final de la revista solo pone mi nombre de autor en el de *Navidad*, en el de *Super Yohsy* pone como autor a un tal Enrique Medina Gremaldos cuando es mío, basta ejecutar el juego para que se vea una pantalla grande en la que pone "Daniel García Alonso presenta". Bueno, sólo era para que lo supierais y, si pudierais, publicar en el siguiente número el error para aclarar quién es el autor ¿Ok? Gracias.
Portox.

Dear Divmanía: acabo de adquirir vuestro sexto número y he podido comprobar vuestra lista de juegos no premiados enviados al concurso. Me ha parecido muy bien. No le ha parecido tan bien a mi amigo Félix, que programó gran parte del juego *Wizfun* y cuyo nombre no aparece entre los creadores. Por el bien de su salud mental rogaría que en el próximo número de su revista incluyan una corrección indicando su nombre (Félix Aurelio Cuadrado Latasa) cómo programador del juego "Wizfun". Gracias y un saludo.
Daniel Fernández. Cuchipandi technologies.

PROGRAMAR JUEGOS ES COSA DE NIÑOS SI TE SUSCRIBES

a Div Manía

Si deseas estar en la vanguardia del mundo de la informática, suscribirse a DIV MANÍA es un primer paso acertado porque...

- *Es la única revista escrita por y para los programadores de videojuegos. Nuestra redacción está compuesta por veteranos desarrolladores, expertos del entorno DIV, grafistas y muchos otros profesionales del software de entretenimiento que dan lo mejor de sí a los lectores.*
- *Te ofrece lo último en el delicado campo de la programación de videojuegos, con los títulos que se encuentran en proceso y los productos recién salidos del horno o de los PCs.*
- *Para seguir avanzando hay que saber echar la vista atrás y a la vez no olvidarse del futuro, y **Div Manía** empieza un nuevo camino.*
- *Nuestra revista presenta un look muy cercano a sus lectores, salido de las inquietudes de todos vosotros.*
- *Nunca nadie te ha ofrecido tanto por tan poco; nunca has tenido tan cerca la oportunidad de estar al día de lo último en programación por el mínimo esfuerzo de acercarte al quiosco o enviar nuestro cupón y recibir la revista en casa puntualmente cada dos meses.*
- *En el interior del CD-Rom encontrarás los elementos con los que todos los programadores sueñan.*
- *Somos como tú y conocemos, más o menos, qué se esconde dentro de tu cabeza. Y si no lo conocemos aún, lo aprenderemos gracias a ti.*
- *Ofrecemos las más diversas sorpresas, las más interesantes ofertas, para que no te olvides de que la programación siempre está viva.*

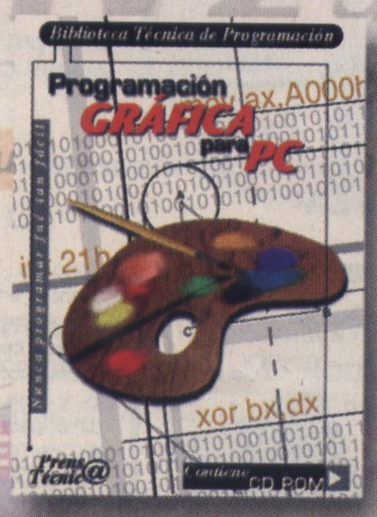
Además, el suscriptor tiene derecho a la siguiente oferta:

Con un año de suscripción (seis números) regalamos un producto a elegir entre:

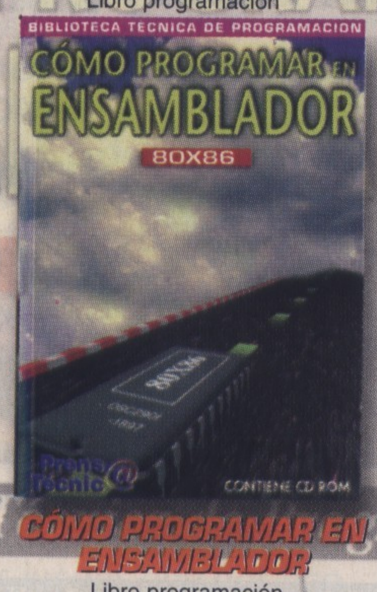
"Programación Gráfica para PC"

"Cómo programar en Ensamblador"

Con dos años de suscripción (doce números) regalamos DIV 2



PROGRAMACIÓN GRÁFICA PARA PC
Libro programación



CÓMO PROGRAMAR EN ENSAMBLADOR
Libro programación

DIV II
Creación de juegos

Prensa Técnica se reserva el derecho de cambiar cualquiera de los regalos por otros de igual valor, en caso de agotar existencias.

Solicite su ejemplar enviando este cupón por correo, por fax: 91 304.17.97 o llamando al teléfono 91 304.06.22 de 9:00 a 19:00 h.

CUPÓN DE SUSCRIPCIÓN ANUAL A DIV MANÍA

Deseo suscribirme a la revista **DIV MANÍA** acogiéndome a la siguiente modalidad:

☐ Suscripción: 1 año (6 números) por sólo 5.970 ptas. ☐ Correo certificado 1 año: 1.500 ptas. adicionales.

☐ Suscripción: 2 año (12 números) por sólo 11.940 ptas. ☐ Correo certificado 2 años: 3.000 ptas. adicionales

Desde el número ☐

Además recibiré gratis:

☐ Por 1 año de suscripción: **uno** de los siguientes productos:
☐ Por 2 años de suscripción: **DIV II**

☐ Programación Gráfica para PC ☐ Cómo programar en Ensamblador

Nombre y apellidos .....
Domicilio .....
Población .....
C.P.... .....Provincia .....
Telf. ..... Profesión .....
DNI/NIF:.....

FORMA DE PAGO:

☐ Con cargo a mi tarjeta VISA nº_____más gastos de envío
Fecha de caducidad de la tarjeta ....
Nombre del titular, si es distinto .....
☐ Domiciliación bancaria, más gastos de envío
Población .....
Ruego a Vd. que se sirva cargar en mi:

☐ cuenta corriente
☐ libreta de ahorro número

ENTIDAD	OFICINA	DC	Nº CUENTA

el recibo que será presentado por PRENSA TÉCNICA S.L. como pago de mi suscripción a la revista **DIV MANÍA** más gastos de envío.

FIRMA:

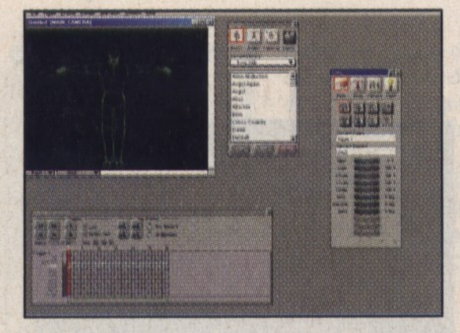
☐ Contrarreembolso del importe más gastos de envío.
☐ Cheque a nombre de PRENSA TÉCNICA, que adjunto más gastos de envío.
☐ Giro Postal (adjunto fotocopia del resguardo) más gastos de envío.

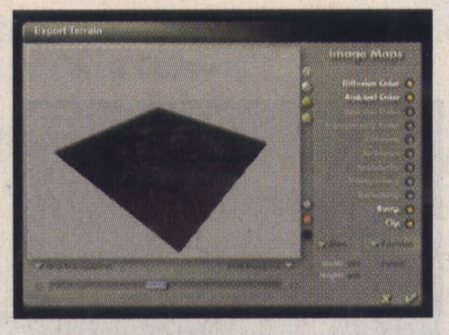
Prens Técnic@
de libros y publicaciones

C/ Alfonso Gómez, nº 42 Nave 1-1-2. 28037 Madrid. Tfno: 91 304 06 22. Fax: 91 304 17 97
e-mail: maspc@prensatecnica.com. http://www.prensatecnica.com

Fénix

Otra estrella de nuestro CD es la versión beta *0.6* de *Fénix* para Windows y para Linux, un fichero con el código fuente y un juego de ejemplo: *La rata y las serpientes*. Como es lógico encontraréis también toda la información necesaria para su instalación y uso que os resumimos aquí a través de las palabras del propio autor: *Fenix* es un lenguaje pseudo-interpretado, diseñado para poder hacer juegos de dos dimensiones (a base de "sprites") fácilmente. Para ello incluye una extensa librería, dedicada a los juegos, que permite programar únicamente la lógica de movimiento de los gráficos y objetos del juego, mientras cosas como la visualización en pantalla de gráficos y sprites, o la reproducción del sonido, corren a cargo del intérprete. Si quieres más información lee el artículo DIV interno de este número de la revista.





CURSOS DE JUEGOS

Todos los complementos que necesitas para seguir todos las lecciones de nuestros cursos. Casi todas las secciones llevan códigos que por su extensión no es posible meter en la revista. En el Cd-Rom los encontrarás enteros y podrás seguir las explicaciones comodamente.

JUEGOS

En próximos números publicaremos otra vez todos los juegos que nos llegan al concurso. Si tenéis algún juego que queréis ansiosamente ver publicado sólo tenéis que enviárnoslo.

Bots Batcher

Un juego de Ferminho, el colaborador de la revista. Para que matéis el rato y acabéis con la próxima invasión alienígena que amenaza, una vez más, al sufrido Planeta Azul. Gracias Ferminho.

Master Mind

Santiago Jiménez Escudero nos ha mandado este sesudo juego para que lo disfrutéis todos vosotros. Se trata de *Master Mind*, un conocido juego de mesa. La finalidad es averiguar la clave secreta que el ordenador da automáticamente. Dicha clave se compone de cinco colores de los ocho posibles que puedes utilizar, pueden ser cinco colores distintos, los cinco colores iguales o colores repetidos, llegando a componer 32.768 permutaciones distintas. Además dispone de un modo dos jugadores. Gracias Santiago.

TEXTURAS Y LIBRERÍAS

Para que podáis utilizarlas en vuestras creaciones lúdicas os metemos unas cuantas texturas y librerías que seguro que serán de utilidad a más de uno.

Registro de Poser 2 y Bryce 2

Para usar correctamente ambos programas tenéis que pedir el número de serie de los mismos llamando al teléfono 906 42 37 58. Una vez obtenido, podéis usar estos dos estupendos programas completos sin ningún coste adicional. Si te interesa actualizarlos lee el cupón adjunto.

Oferta para los Lectores de DIV manía

Atlantic Devices, distribuidor de productos de Metacreations en España, ofrece una interesante oferta de descuento para aquellos lectores que, teniendo la versión de Bryce 2 y Poser 2 que regalamos en nuestro CD-ROM especial, deseen actualizarse a la versión 4 de estas aplicaciones.

Sí, deseo acogerme a la oferta de descuento de...

☐ 3.000 pesetas en la actualización de Poser 4.

☐ 3.000 pesetas en la actualización de Bryce 4.

Nombre ..... Domicilio .....Teléfono .....

N.I.F.: ..... Población ..... Provincia ..... Código Postal .....

FORMA DE PAGO

☐ Con cargo a mi tarjeta VISA Nº. .....

Fecha de caducidad de la tarjeta..... Nombre del titular, si es distinto.....

☐ Domiciliación bancaria

Población.....

Ruego a vd. Que se sirva cargar en mi

☐ Cuenta Corriente

☐ Libreta de ahorro número

ENTIDAD	OFICINA	DC	Nº CUENTA

el recibo que será presentado por ATLANTIC DEVICES como pago de la adquisición del software elegido con su oferta correspondiente.

☐ Contra Reembolso del importe más gastos de envío

☐ Cheque a nombre de ATLANTIC DEVICES, que adjunto.

☐ Giro Postal (adjunto fotocopia del resguardo).

*No se admitirán fotocopias de cupones ni cupones enviados por Fax.

EL CUPÓN DEBE ENVIARSE A LA SIGUIENTE DIRECCIÓN:
ATLANTIC DEVICES
C/ Caputxins, 58
08700 Igualada (Barcelona)

Tenemos todo lo que buscas

Prens@ Técnic
de publicaciones y libros

- Prensa Técnica te ofrece los últimos avances y novedades del mundo de la informática a través de sus publicaciones.
- Internet, Linux, Diseño digital, Programación, Juegos... una oferta variadísima que cubre todo lo que necesitas para estar al día.
- Tenemos revistas para todos los públicos, ya seas principiante o avanzado, Prensa Técnica tiene la solución a tus problemas.

¡Más de 350.000 lectores cada mes!

LA REVISTA QUE TE DA MÁS

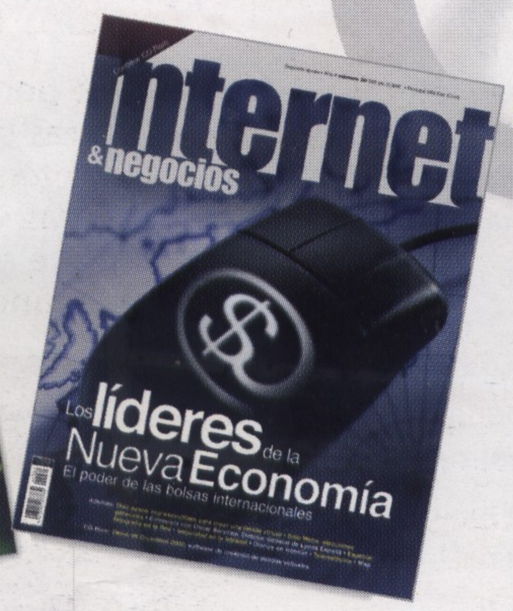
MÁS PC, la revista informática para todos los públicos, con toda la información y actualidad en hardware, software, Internet, diseño, Linux, programación, videojuegos, multimedia, etc.

Pc
Incluye CD-Rom y libro técnico

TU ORDENADOR AL DÍA

CD DRIVER es una revista imprescindible para el mantenimiento de tu PC. Con ella, el usuario informático tendrá a mano todos los drivers del mercado y estupendos artículos sobre la utilización e instalación de los componentes del PC.

Pc
Bimestral *Incluye 2 CD-Roms*



TU GUÍA PARA LA RED

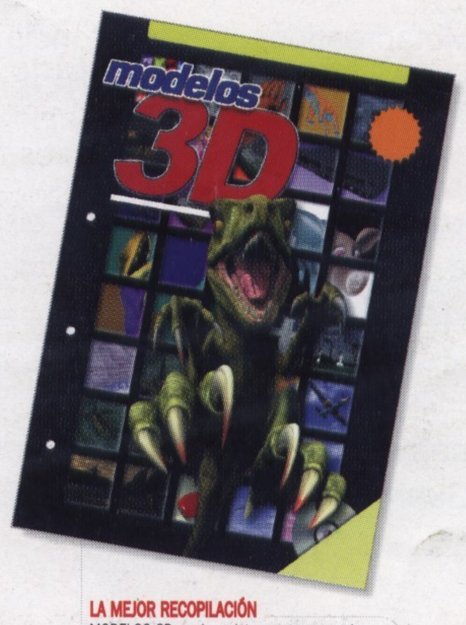
INTERNET Y NEGOCIOS se introduce en los recovecos de la Red mostrándole información rigurosa sobre aspectos técnicos, análisis de webs y herramientas. Incluye CD-Rom con navegadores, utilidades de correo, chat, etc.

Pc • Mac
Incluye CD-Rom

LA MÁS VENDIDA DE EUROPA

ELECTRÓNICA PRÁCTICA ACTUAL es la edición en castellano de la revista de electrónica más vendida de Europa. Contenidos prácticos de electrónica e informática con noticias, Internet y los montajes más ingeniosos.

Pc
Incluye CD-Rom



LA MEJOR RECOPILACIÓN

MODELOS 3D es la revista que te proporciona todos los modelos y texturas que necesitas sin tener que perder el tiempo buscándolas. Incluye modelos, texturas y demos de los programas 3D más utilizados.

Pc • Mac
Bimestral *Incluye CD-Rom*

PURO JAVA PARA TODOS

JAVA MAGAZINE es una nueva revista que te sorprenderá, adetrándote en los secretos de este lenguaje de programación y las técnicas más usadas del momento. Todo a un mundo a tu alcance de la mano del apasionante Java

Pc
Incluye CD-Rom

CREAR ESTÁ EN TUS MANOS

3D WORLD está especializada en infografía y en general las 3D. Con la última actualidad en diseño gráfico, reportajes, técnicas, trucos y tutoriales de los programas de diseño y 3D más utilizados en el sector profesional.

Pc • Mac
Incluye CD-Rom

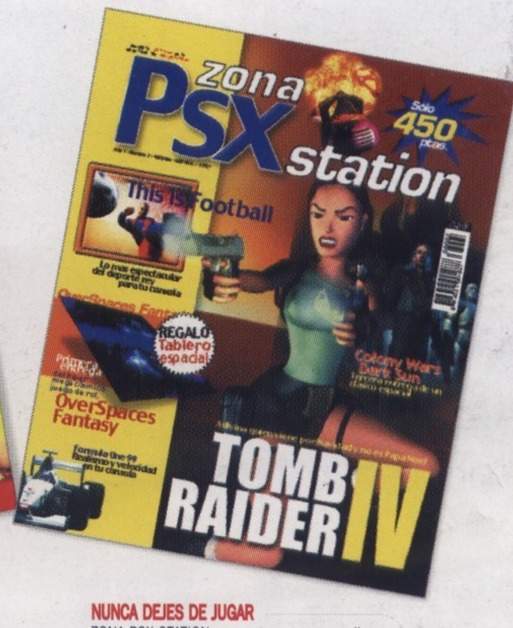
LA NUEVA ERA DE LA FOTOGRAFÍA Y EL ARTE

FOTO ACTUAL Y ARTE DIGITAL, revista para profesionales y aficionados al diseño, maquetación y retoque fotográfico. La mejor forma de conocer toda la teoría y la práctica sobre las técnicas más utilizadas del momento.

Pc • Mac
Incluye CD-Rom

JUGANDO DURO

N-ZONE te informa de todas las novedades del sistema Nintendo. De la consola Nintendo 64, Game Boy y Game Boy Color. Es la primera revista en informar sobre las novedades y los juegos que están por salir al mercado.



NUNCA DEJES DE JUGAR

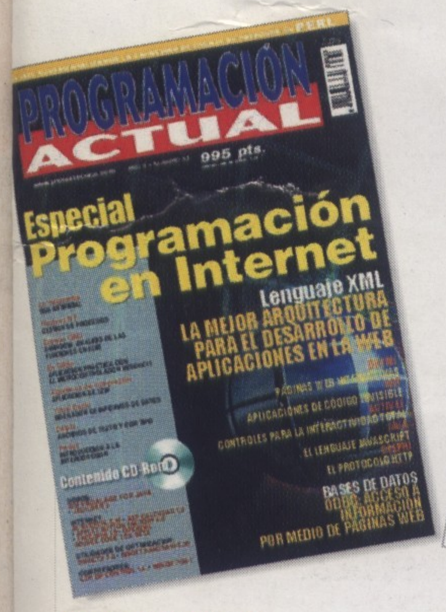
ZONA PSX STATION encuentra una nueva dimensión para tu Playstation con una revista llena de originales secciones, objetiva y con un diseño que da a las imágenes la importancia que se merecen.

Incluye suplemento Xtreme PSX

HAZ TUS PROPIOS VIDEOJUEGOS

DIV MANÍA es la primera revista dedicada a aprender a programar videojuegos, abarcando todos los aspectos del desarrollo. Incluye CD-Rom con tres juegos programados por los lectores y demos de juegos profesionales.

Pc
Bimestral *Incluye CD-Rom*



POR Y PARA PROGRAMADORES

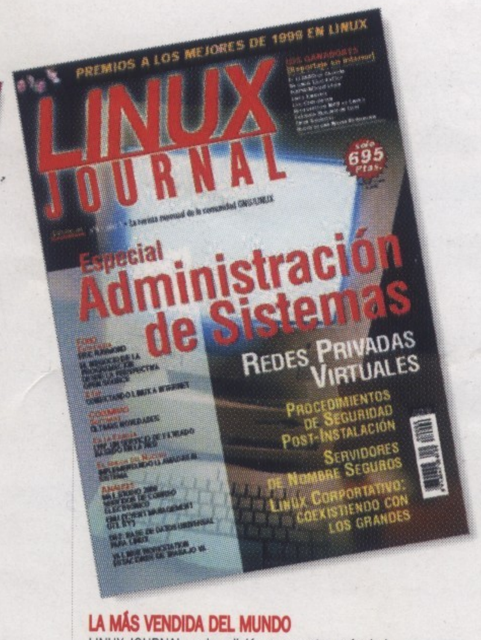
PROGRAMACIÓN ACTUAL te pone al día del mundo del desarrollo gracias a sus secciones principales dedicadas a la programación gráfica, Internet y sus lenguajes, desarrollo empresarial y nuevas tecnologías.

Pc
Incluye CD-Rom

LO ÚLTIMO EN TECNOLOGÍA

WINDOWS NT ACTUAL está destinada a profesionales del mundo NT. El modo más fácil para estar al día y conocer el entorno NT así como sus aplicaciones.

Pc
Incluye CD-Rom



LA MÁS VENDIDA DEL MUNDO

LINUX JOURNAL es la edición en nuestro país de la publicación más prestigiosa del mundo GNU/Linux. Entrevistas, actualidad y buenos artículos se dan cita en una auténtica "BIBLIA" sobre este sistema operativo.

LO MEJOR, AHORA EN CASTELLANO

LINUX ACTUAL es la primera revista en castellano dedicada al GNU/Linux: el sistema operativo de moda. Incluye artículos dedicados a todas sus áreas y un CD-Rom con las mejores distribuciones y novedades del momento.

Pc
Bimestral *Incluye CD-Rom*

PENSADA PARA PRINCIPIANTES

SÓLO LINUX es la mejor revista en castellano para el usuario principiante en el mundo GNU/Linux. En ella encuentra toda la información en forma de artículos de nivel básico. Incluye un CD-Rom con la distribución más fácil de instalar del momento.

Pc
Bimestral *Incluye CD-Rom*

Edita **PRENSA TÉCNICA** de libros y publicaciones • Alfonso Gómez 42, Nave 1-1-2. • 28037 Madrid • Tf: 91 3.04.06.22 • Fax: 91 3.04.17.97 • http://www.prensatecnica.com

CONTENIDO DEL CD ROM

HERRAMIENTAS PARA TUS JUEGOS

Ponemos a disposición de los lectores de DIVmanía dos programas totalmente completos y operativos, una demo de actualidad y la versión beta del programa *Fénix*. Seguro que sabrás sacarle el jugo al contenido de este CD-Rom.

POSER 2
Este programa es una completa herramienta de creación y animación de personajes. Imprescindible para tus juegos.

BRYCE 2
Uno de los más potentes generadores de entornos tridimensionales del mercado. Para crear paisajes donde meter a tus personajes.

DEMO BRYCE 4
Lo último de lo último de Metacreations. Gran potencia y sencillez de manejo son las virtudes de la última versión de *Bryce*.

FÉNIX
Este programa es un lenguaje pseudo-interpretado, diseñado para poder hacer juegos de dos dimensiones de forma fácil y sencilla, tanto para Windows como para Linux.

JUEGOS GANADORES
Los afortunados de este mes han sido: *Preludio*, primer premio; *99-00*, en segundo lugar; y *¡Guerra!* en el tercer puesto. Enhorabuena a los tres ganadores.

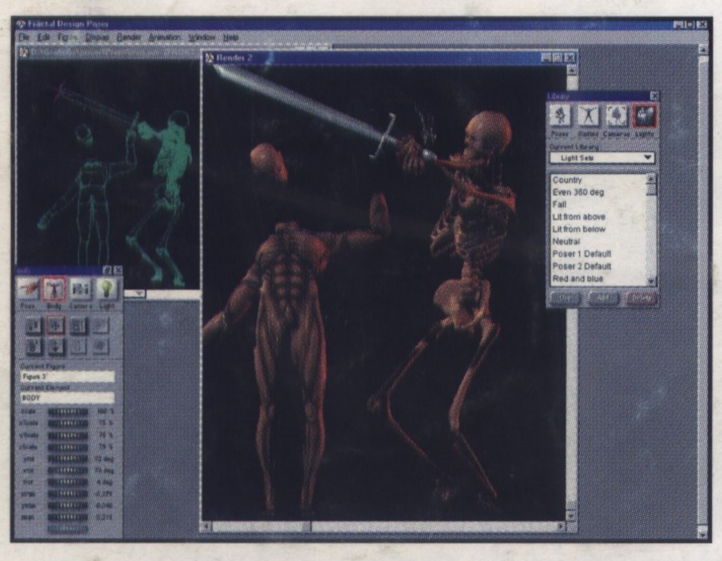
JUEGOS DE REGALO
Bots Batcher, un matamarcianos y *Master Mind*, conversión electrónica del conocido juego de mesa.

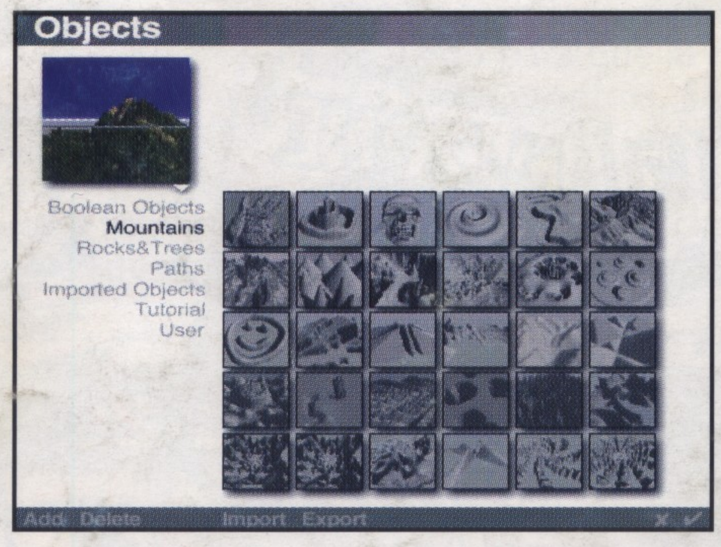
Cursos de juegos
Todos los complementos que necesitas para seguir todos las lecciones de nuestros cursos. Casi todas las secciones llevan códigos que por su extensión no es posible meter en la revista, pero podrás encontrarlos en el CD-Rom.

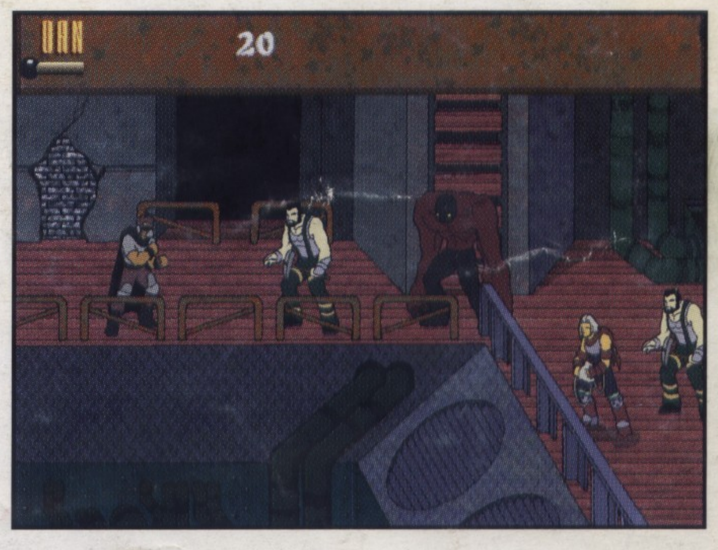
REPORTAJES. Lo que debes saber sobre *Poser* y la animación de personajes.

PROGRAMAS. Herramientas para diseñar los escenarios para vuestros juegos.

CONCURSO DE JUEGOS. Los ganadores de nuestro reñido concurso de videojuegos.







DIVmanía CON EL MEJOR CONTENIDO

ACTUAL EXHAUSTIVO DIDÁCTICO

Y MUCHO MÁS...